c h p t e r operate system software manage computer hardware provide basis application program act intermediary computer user computer hardware amazing aspect operating system vary accomplish task wide variety compute environment operate system car home appliance include internet things device smart phone personal computer enterprise computer cloud computing envi- order explore role operate system modern compute environment important understand organization architec- ture computer hardware include cpu memory o device storage fundamental responsibility operate system allocate resource program operate system large complex create piece piece piece delineate portion system carefully define input output function chapter provide general overview major component contemporary computer system function provide operate system additionally cover topic help set stage remainder text data structure operate system compute environment open source free operating system describe general organization computer system role describe component modern multiprocessor computer system illustrate transition user mode kernel mode discuss operating system computing environ- provide example free open source operating system operating system begin discussion look operate system role overall computer system computer system divide roughly component hardware operating system application program user figure 1.1 hardware central processing unit cpu memory input output o device provide basic compute resource system application program word processor spreadsheet compiler web browser define way resource solve user computing problem operate system control hardware coordinate use application program user view computer system consist hardware software datum operate system provide mean proper use resource operation computer system operate system similar government like government perform useful function simply provide environment program useful work understand fully operate system role explore operate system viewpoint user system user view computer vary accord interface computer user sit laptop pc consist monitor keyboard mouse system design user monopolize resource goal maximize work play user perform case operate system design ease use attention pay performance security pay operate system design ease use attention pay performance security pay resource utilization hardware software resource compiler web browser development kit etc cpu memory o device etc abstract view component computer system operating system increasingly user interact mobile device smartphone tablet device replace desktop laptop computer system user device typically connect network cellular wireless technology user interface mobile computer generally feature touch screen user interact system press swipe finger screen physical keyboard mouse mobile device allow user interact voice recognition interface apple siri computer little user view example embed com- puter home device automobile numeric keypad turn indicator light status operate sys- tem application design primarily run user intervention computer point view operate system program intimately involve hardware context view oper- ating system resource allocator computer system resource require solve problem cpu time memory space storage space o device operate system act manager resource face numerous possibly conflicting request resource operate system decide allocate specific program user operate computer system efficiently fairly slightly different view operate system emphasize need control o device user program operate system control program control program manage execution user program prevent error improper use computer especially concern operation control o device define operating systems probably term operating system cover role function case myriad design use computer computer present toaster car ship spacecraft home business basis game machine cable tv tuner industrial control system explain diversity turn history computer computer relatively short history evolve rapidly comput- e start experiment determine quickly move fix purpose system military use code breaking trajectory plotting governmental use census calculation early computer evolve general purpose multifunction mainframe operating system bear 1960s moore law predict number transistor integrate circuit double 18 month prediction hold true computer gain functionality shrink size lead vast number use vast number variety operate system appendix detail history operating system vast number variety operate system appendix detail history operating system define operate system general completely adequate definition operate system operate system exist offer reasonable way solve problem create usable compute system fundamental goal computer system execute program solve user problem easy computer hardware construct goal bare hardware particularly easy use application program develop program require certain common operation control o device common function control allocate resource bring piece software operate system addition universally accept definition operate system simple viewpoint include ven- dor ship order operating system feature include how- vary greatly system system megabyte space lack screen editor require gigabyte space base entirely graphical windowing system com- mon definition usually follow operate system program run time computer usually call kernel kernel type program system program associate operate system neces- sarily kernel application program include program associate operation system matter constitute operate system increasingly important personal computer widespread operate sys- tem grow increasingly sophisticated 1998

united states department justice file suit microsoft essence claim microsoft include functionality operate system prevent application vendor compete example web browser integral microsoft operate system result microsoft find guilty operating system monopoly limit competition today look operating system mobile device number feature constitute operate system increase mobile operate system include core kernel middleware set software framework provide additional service application developer example promi- nent mobile operating system apple ios google android feature study operating system practitioner computer science small per- centage involve creation modification operat- e system study operate system work simply code run operate system knowledge operate system work crucial proper efficient effective secure programming understand fundamental operate system drive computer hardware provide application essential program highly useful write program use core kernel middleware support database multimedia graphic summary purpose operate system include always- run kernel middleware framework ease application development provide feature system program aid manage system run text concern kernel general- purpose operate system component discuss need fully explain operating system design operation modern general purpose computer system consist cpu number device controller connect common bus provide access component share memory figure 1.2 device controller charge specific type device example disk drive audio device graphic display depend controller device attach instance system usb port connect usb hub device connect device controller maintain local buffer storage set special purpose register device controller responsible move datum peripheral device control local buffer storage typically operate system device driver device con- troller device driver understand device controller provide rest operate system uniform interface device cpu device controller execute parallel compete memory cycle ensure orderly access shared memory memory controller synchronize access memory follow subsection describe basic system operate focus key aspect system start interrupt alert cpu event require attention discuss storage structure o structure typical pc computer system consider typical computer

structure o structure typical pc computer system consider typical computer operation program perform o. start o operation device driver load appropriate register device controller device controller turn examine content reg- ister determine action read character keyboard controller start transfer datum device local buffer transfer datum complete device controller inform device driver finish operation device driver give control part operate system possibly return datum pointer datum operation read operation device driver return status information write complete successfully device busy controller inform device driver finish operation accomplish interrupt hardware trigger interrupt time send signal cpu usually way system bus bus computer system system bus main communication path major component interrupt purpose key operate system hardware interact transfer execution fix location fix location usually contain start address service routine interrupt locate interrupt service routine execute completion cpu resume interrupt computation timeline operation show figure 1.3 run animation assicate figure click interrupt important computer architecture computer design interrupt mechanism function common interrupt transfer control appropriate interrupt service routine straightforward method manage transfer invoke generic routine examine interrupt information routine turn interrupt specific handler interrupt handle quickly occur frequently table pointer interrupt routine instead provide necessary speed interrupt routine call indirectly table intermediate routine need generally table pointer store low memory location location hold address interrupt service routine device array interrupt vector address index unique number give interrupt request provide address interrupt service routine interrupt device operate system different windows unix dispatch interrupt manner interrupt architecture save state information interrupt restore information service interrupt interrupt routine need modify processor state instance modify register value explicitly save current state restore state return interrupt service save return address load program counter interrupt computation resume interrupt occur basic interrupt mechanism work follow cpu hardware wire call interrupt request line cpu sense execute instruction cpu detect controller assert signal

line cpu sense execute instruction cpu detect controller assert signal interrupt request line read interrupt number jump interrupt handler routine interrupt number index interrupt vector start execution address associate index interrupt handler save state change operation determine cause interrupt perform necessary processing perform state restore execute return interrupt instruction return cpu execution state prior interrupt device controller raise interrupt assert signal interrupt request line cpu catch interrupt dispatch interrupt handler handler clear interrupt service device figure 1.4 summarize interrupt drive o cycle basic interrupt mechanism describe enable cpu respond asynchronous event device controller ready service modern operate system need sophisticated interrupt- 1 need ability defer interrupt handling critical processing 2 need efficient way dispatch proper interrupt handler 3 need multilevel interrupt operate system distin- guish high- low priority interrupt respond appropriate degree urgency modern computer hardware feature provide cpu interrupt controller hardware device driver initiate o cpu receive interrupt transfer control cpu execute check interrupt instruction return interrupt input ready output complete error generate interrupt signal interrupt drive o cycle cpu interrupt request line nonmaskable interrupt reserve event unrecoverable memory error second interrupt line maskable turn cpu execution critical instruction sequence interrupt maskable interrupt device controller request service recall purpose vectored interrupt mechanism reduce need single interrupt handler search possible source interrupt determine need service practice computer device interrupt handler address element interrupt vector common way solve problem use interrupt chaining element interrupt vector point head list interrupt handler interrupt raise handler correspond list call find service request structure compromise overhead huge interrupt table inefficiency dispatch single interrupt handler figure 1.5 illustrate design interrupt vector intel processor event 0 31 nonmaskable signal error condition event 32 255 maskable purpose device generate interrupt interrupt mechanism implement system interrupt priority level level enable cpu defer handling low priority inter- bound range exception

enable cpu defer handling low priority inter- bound range exception device available coprocessor segment overrun reserved invalid task state segment segment present intel reserved use intel reserved use intel processor event vector table rupt mask interrupt make possible high priority interrupt preempt execution low priority interrupt summary interrupt modern operate system handle asynchronous event purpose discuss through- text device controller hardware fault raise interrupt enable urgent work modern computer use system interrupt priority interrupt heavily time sensitive processing efficient interrupt handling require good system perfor- cpu load instruction memory program load memory run general purpose computer run program rewritable memory call main memory call random access memory ram main memory commonly implement semiconductor technology call dynamic random access memory dram computer use form memory example pro- gram run computer power bootstrap program load operate system ram volatile lose content power turn lose trust hold bootstrap pro- gram instead purpose computer use electri- cally erasable programmable read memory eeprom form firmwar storage infrequently write nonvolatile eeprom storage definition notation basic unit computer storage bit bit contain value 0 1 storage computer base collection bit give bit amazing thing computer represent number letter image movie sound document program byte 8 bit computer small convenient chunk storage example computer instruction bit byte common term word give computer architecture native unit datum word byte example computer 64 bit register 64 bit memory addressing typically 64 bit 8 byte word computer execute operation native word size byte time computer storage computer throughput generally measure manipulate byte collection byte kilobyte kb 1,024 byte megabyte mb 1,0242 byte gigabyte gb 1,0243 byte terabyte tb 1,0244 byte petabyte pb 1,0245 byte computer manufacturer round number megabyte 1 million byte gigabyte 1 billion byte networking measurement exception general rule give bit network datum bit time change change frequently addition

give bit network datum bit time change change frequently addition low speed contain static program datum frequently example iphone use eeprom store serial number hardware information device form memory provide array byte byte address interaction achieve sequence load store instruc- tion specific memory address load instruction move byte word main memory internal register cpu store instruction move content register main memory aside explicit load store cpu automatically load instruction main memory execution location store program counter typical instruction execution cycle execute system von neumann architecture fetch instruction memory store instruction instruction register instruction decode cause operand fetch memory store internal register instruction operand execute result store memory notice memory unit see stream memory address know generate instruction counter indexing indirection literal address mean instruction datum accordingly ignore memory address generate program interested sequence memory address generate run program ideally want program datum reside main memory per- manently arrangement usually possible system 1 main memory usually small store need program datum 2 main memory mention volatile lose content power turn lose computer system provide secondary storage extension main memory main requirement secondary storage able hold large quantity datum permanently common secondary storage device hard disk drive hdds nonvolatile memory nvm device provide storage program datum program system application store secondary storage load memory program use secondary storage source destination processing secondary storage slow main memory proper management secondary storage central importance computer sys- tem discuss chapter 11 large sense storage structure describe consist register main memory secondary storage possible storage system design possible component include cache memory cd rom blu ray magnetic tape slow large special purpose store backup copy material store device example call tertiary storage storage system provide basic function store datum hold datum retrieve later time main difference storage system lie speed size wide variety storage system organize

storage system lie speed size wide variety storage system organize hierarchy figure 1.6 accord storage capacity access time general rule trade size speed small fast memory close cpu show figure addition differ speed capacity storage system volatile nonvolatile volatile storage mention early lose content power device remove datum write nonvolatile storage safekeeping level memory figure construct semi- conductor memory consist semiconductor base electronic circuit nvm device fourth level variant general fast hard disk common form nvm device flash memory popular mobile device smartphone tablet increasingly flash memory long term storage laptop desktop server storage play important role operating system structure refer frequently text general use follow volatile storage refer simply memory need empha- size particular type storage device example register),we nonvolatile storage retain content power lose refer nvs vast majority time spend nvs secondary storage type storage classify

mechanical example storage system hdds optical disk holographic storage magnetic tape need emphasize particular type mechanical storage device example magnetic tape explicitly

electrical example storage system flash memory fram nram ssd electrical storage refer nvm need emphasize particular type electrical storage device example ssd explicitly mechanical storage generally large expensive byte electrical storage conversely electrical storage typically costly small fast mechanical storage design complete storage system balance factor discuss use expensive memory necessary provide inexpensive nonvolatile storage possible cache instal improve performance large disparity access time transfer rate exist component large portion operating system code dedicate manage o importance reliability performance system vary nature device recall beginning section general purpose computer system consist multiple device exchange datum common thread execution modern computer system work bus form interrupt drive o describe section 1.2.1 fine move small amount datum produce high overhead bulk datum movement nvs o. solve problem direct memory access dma set buffer pointer counter o device device controller transfer entire block datum directly device main memory intervention cpu interrupt generate block tell device driver operation complete interrupt byte generate low speed device device controller perform operation cpu available accomplish work high end system use switch bus architecture system multiple component talk component concurrently compete cycle shared bus case dma effective figure 1.7 show interplay component computer section 1.2 introduce general structure typical computer sys- tem computer system organize number different way categorize roughly accord number general purpose year ago computer system single processor contain cpu single processing core core component exe- cut instruction register store datum locally main cpu core capable execute general purpose instruction set include instruction process system special purpose proces- sor come form device specific processor disk keyboard graphic controller special purpose processor run limited instruction set run process manage operate system operate system send information task monitor status example disk controller microprocessor receive sequence request main cpu core implement disk queue scheduling algorithm arrangement relieve main cpu

core implement disk queue scheduling algorithm arrangement relieve main cpu overhead disk scheduling pc contain microprocessor keyboard convert keystroke code send cpu system circumstance special purpose processor low level component build hardware operate system communicate proces- sor job autonomously use special purpose microproces- sor common turn single processor system multiproces- sor general purpose cpu single processing core system single processor system accord definition contemporary computer system single processor system modern computer mobile device server multiprocessor sys- tem dominate landscape computing traditionally system processor single core cpu proces- sor share computer bus clock memory periph- eral device primary advantage multiprocessor system increase throughput increase number processor expect work time speed ratio n processor n n.

multiple processor cooperate task cer- tain overhead incur keep part work correctly overhead plus contention share resource lower expect gain additional processor common multiprocessor system use symmetric multiprocess- e smp peer cpu processor perform task include operating system function user process figure 1.8

illustrate typical smp architecture processor cpu notice cpu processor set register private local cache processor share physical memory system bus benefit model process run simultaneously n process run n cpu cause performance deteriorate significantly cpu separate sit idle overloaded result inefficiency inefficiency avoid processor share certain data structure multiprocessor system form allow process resource memory share dynamically processor lower workload variance processor system write carefully shall chapter 5 chapter 6 definition multiprocessor evolve time include multicore system multiple compute core reside single chip multicore system efficient multiple chip single core chip communication fast chip communication symmetric multiprocessing architecture addition chip multiple core use significantly power multiple single core chip important issue mobile device figure 1.9 dual core design core pro- cessor chip design core register set local cache know level 1 l1 cache notice level 2 l2 cache local chip share processing core archi- tecture adopt approach combine local share cache local low level cache generally small fast high level shared dual core design core chip definition computer system component cpu hardware execute instruction processor physical chip contain cpu core basic computation unit cpu multicore include multiple compute core cpu multiprocessor include multiple processor virtually system multicore use general term cpu refer single computational unit computer system core multicore specifically refer core cache aside architectural consideration cache memory bus contention multicore processor n core appear operate sys- tem n standard cpu characteristic put pressure operating system designer application programmer efficient use pro- cesse core issue pursue chapter 4 virtually modern operate system include windows macos linux android ios mobile system support multicore smp system add additional cpu multiprocessor system increase comput- e power suggest early concept scale add cpu contention system bus bottleneck performance begin degrade alternative approach instead provide cpu group cpu local memory access small fast local bus

cpu group cpu local memory access small fast local bus cpu connect share system interconnect cpu share physical address space approach know non uniform memory access numa illustrate figure 1.10 advantage cpu access local memory fast contention system interconnect numa system scale effectively processor add apotential drawback numasystem increase latency cpu access remote memory system interconnect create possible performance penalty word example cpu0 access local memory cpu3 quickly access local memory slow performance operate system minimize numa penalty careful cpu scheduling memory management discuss section 5.5.2 section 10.5.4 numa system scale accommodate large number processor increasingly popular server high performance computing system finally blade server system multiple processor board o board networking board place chassis differ- ence traditional multiprocessor system blade- processor board boot independently run operating system blade server board multiprocessor blur line numa multiprocessing architecture type computer essence server consist multiple independent type multiprocessor system cluster system gath- er multiple cpu clustered system differ multiprocessor system describe section 1.3.2 compose individual system node join node typically mul- ticore system system consider loosely couple note definition cluster concrete commercial open- source package wrestle define cluster system form well generally accept definition clustered computer share storage closely link local area network lan describe chapter 19 fast interconnect infiniband clustering usually provide high availability service service continue system cluster fail generally obtain high availability add level redundancy system layer cluster software run cluster node node monitor network monitor machine fail monitor machine ownership storage restart application run fail machine user client application brief interruption service high availability provide increase reliability crucial application ability continue provide service proportional level survive hardware call graceful degradation system graceful degradation call fault tolerant suffer failure single component continue operation fault tolerance require mechanism allow failure detect diagnose cluster structure asymmetrically symmetrically asym- metric clustering machine

detect diagnose cluster structure asymmetrically symmetrically asym- metric clustering machine hot standby mode run- ne application hot standby host machine monitor active server server fail hot standby host active consider desktop pc motherboard processor socket show board fully function computer slot populate consist processor socket contain cpu dram socket pcie bus slot o connector type low cost general- purpose cpu contain multiple core motherboard contain multiple processor socket advanced computer allow system board create numa system server symmetric clustering host run application monitor structure obviously efficient use available hardware require application available run cluster consist computer system connect net- work cluster provide high performance computing envi- ronment system supply significantly great computational power single processor smp system run application concurrently computer cluster application write specifically advantage cluster involve technique know parallelization divide program separate component run parallel individual core computer comput- er cluster typically application design compute node cluster solve portion problem result node combine final solution form cluster include parallel cluster cluster wide area network wan describe chapter 19 parallel cluster allow multiple host access datum share storage oper- general structure cluster system ate system lack support simultaneous datum access multiple host parallel cluster usually require use special version software special release application example oracle real application cluster version oracle database design run parallel cluster machine run oracle layer software track access share disk machine access datum database provide share access system supply access control locking ensure conflict operation occur function commonly know distribute lock manager dlm include cluster technology cluster technology change rapidly cluster product support thousand system cluster cluster node separate mile improvement possible storage area network sans describe section 11.7.4 allow system attach pool storage application datum store san cluster software assign application run host attach san host fail host database cluster dozen host share database

san host fail host database cluster dozen host share database greatly increase performance reliability figure 1.11 depict general structure cluster system discuss basic information computer system organi- zation architecture ready talk operate system oper- ating system provide environment program execute internally operate system vary greatly organize different line commonality consider computer start run instance power reboot need initial program run note early initial program bootstrap program tend simple typically store computer hardware firmware initialize aspect system cpu register device controller memory content bootstrap program know load operate system hadoop open source software framework distribute processing large datum set know big datum cluster system con- taine simple low cost hardware component hadoop design scale single system cluster contain thousand compute node task assign node cluster hadoop arrange communica- tion node manage parallel computation process coalesce result hadoop detect manage failure node provide efficient highly reliable distribute computing service hadoop organize following component distribute file system manage datum file distribute com- yarn resource negotiator framework manage resource cluster schedule task node mapreduce system allow parallel processing datum node cluster hadoop design run linux system hadoop application write programming language include scripting language php perl python java popular choice develop hadoop application hadoop java library support mapreduce information mapreduce hadoop find https://hadoop.apache.org/docs/r1.2.1/mapred tutorial.html start execute system accomplish goal bootstrap program locate operate system kernel load memory

kernel load execute start provide service system user service provide outside kernel system program load memory boot time system daemon run entire time kernel run linux system program systemd start daemon phase complete system fully boot system wait event occur process execute o device service user respond operate system sit quietly wait happen event signal occurrence interrupt section 1.2.1 describe hardware interrupt form interrupt trap exception software generate interrupt cause error example division zero invalid memory access specific request user program operating system service perform execute special operation call system multiprogramming multitasking important aspect operate system ability run multiple program single program general cpu o device busy time furthermore user typically want run program time multiprogramming increase cpu utilization keep user satisfied organize program cpu execute multiprogrammed system program execution term process idea follow operate system keep process memory simultaneously figure 1.12 operate system pick begin execute process eventually process wait task o operation complete non multiprogrammed system cpu sit idle multiprogrammed system operate system simply switch execute process process need wait cpu switch process eventually process finish wait get cpu long process need execute cpu idle idea common life situation lawyer work client time example case wait trial paper type lawyer work case client lawyer idle lack work idle lawyer tend politician certain social value keep lawyer busy multitasking logical extension multiprogramming multitaske system cpu execute multiple process switch switch occur frequently provide user fast response time consider process execute typically execute short time finish need perform o. o

interactive output go display user input come user keyboard mouse touch screen interactive o typically run peo- ple speed long time complete input example memory layout multiprogramme system bound user typing speed seven character second fast people incredibly slow computer let cpu sit idle interactive input take place operate system rapidly switch cpu process have process memory time require form memory management cover chapter 9 chapter 10 addition process ready run time system choose process run make decision cpu scheduling discuss chapter 5 finally run multiple process concur- rently require ability affect limit phase operate system include process scheduling disk storage memory management discuss consideration text multitaske system operate system ensure reasonable nique allow execution process completely memory chapter 10 main advantage scheme enable user run program large actual physical memory abstract main memory large uniform array storage separate logical mem- ory view user physical memory arrangement free programmer concern memory storage limitation multiprogramme multitaske system provide file sys- tem chapter 13 chapter 14 chapter 15 file system reside secondary storage storage management provide chapter 11 addition system protect resource inappropriate use chapter 17 ensure orderly execution system provide mechanism process synchronization communication chapter 6 chapter 7 ensure process stick deadlock forever wait chapter 8) dual mode multimode operation operate system user share hardware software resource computer system properly design operate system ensure incorrect malicious program cause program operate system execute incorrectly order ensure proper execution system able distinguish execution operating system code user define code approach take computer system provide hardware support allow differentiation mode execution need separate mode operation user mode kernel mode call supervisor mode system mode privileged mode bit call mode bit add hardware computer indicate current mode kernel 0 user 1 mode bit distinguish task execute behalf operate system execute behalf user computer system execute

execute behalf operate system execute behalf user computer system execute behalf user application system user mode user application request service operate system system system transition user kernel mode fulfill user process execute call system return system mode bit = 1 mode bit = 0 mode bit = 1 mode bit = 0 execute system transition user kernel mode request show figure 1.13 shall architectural enhancement useful aspect system operation system boot time hardware start kernel mode operate system load start user application user mode trap interrupt occur hardware switch user mode kernel mode change state mode bit 0 operate system gain control computer kernel mode system switch user mode set mode bit 1 pass control user program dual mode operation provide mean protect operate system errant user errant user accomplish protection designate machine instruction cause harm privileged instruction hardware allow privi- leged instruction execute kernel mode attempt execute privileged instruction user mode hardware execute instruction treat illegal trap operate system instruction switch kernel mode example privileged instruction example include o control timer management interrupt management additional privileged instruction discuss text concept mode extend mode example intel processor separate protection ring ring 0 kernel mode ring 3 user mode ring 1 2 vari- ous operating system service practice rarely armv8 system seven mode cpu support virtualization section 18.1 frequently separate mode indicate virtual machine manager vmm control system mode vmm privilege user process few kernel need level privilege create manage virtual machine change cpu state well understand life cycle instruction execution computer system initial control reside operate system instruc- tion execute kernel mode control give user applica- tion mode set user mode eventually control switch operate system interrupt trap system contem- porary operate system microsoft windows unix linux advantage dual mode feature provide great protection operate system system call provide mean

feature provide great protection operate system system call provide mean user program ask operate system perform task reserve operate system user pro- gram behalf system invoke variety way depend functionality provide underlie processor form method process request action operate system system usually take form trap specific location interrupt vector trap execute generic trap instruction system specific syscall instruction invoke system system execute typically treat hardware software interrupt control pass interrupt vector service routine operate system mode bit set kernel mode system service routine operate system kernel exam- ine interrupt instruction determine system occur parameter indicate type service user program request additional information need request pass register stack memory pointer memory location pass reg- ister kernel verify parameter correct legal execute request return control instruction follow system describe system call fully section 2.3 hardware protection place detect error violate mode error normally handle operate system user program fail way make attempt execute illegal instruction access memory user address space hardware trap operate system trap transfer control interrupt vector operate system interrupt program error occur operate system terminate program abnormally situation handle code user request abnormal termination appropriate error message give memory program dump memory dump usually write file user programmer examine correct restart ensure operate system maintain control cpu allow user program stick infinite loop fail system service return control operate system accomplish goal use timer timer set interrupt computer specify period

period fix example 1/60 second variable example 1 millisecond 1 second variable timer generally implement fix rate clock counter operate system set counter time clock tick counter decremente counter reach 0 interrupt occur instance 10 bit counter 1 millisecond clock allow interrupt interval 1 millisecond 1,024 millisecond step 1 millisecond turn control user operate system ensure timer set interrupt timer interrupt control transfer automati- cally operate system treat interrupt

timer interrupt control transfer automati- cally operate system treat interrupt fatal error linux system kernel configuration parameter hz specify fre- quency timer interrupt hz value 250 mean timer generate 250 interrupt second interrupt 4 millisecond value hz depend kernel configure machine type architecture run related kernel variable jiffy represent number timer interrupt occur system boot programming project chapter 2 explore timing linux kernel program time clearly instruction modify content timer privileged see operate system resource manager system cpu memory space file storage space o device resource operate system manage program instruction execute cpu program execution mention process aprogram compiler process word processing program run individual user pc process similarly social medium app mobile device process consider process instance program execution later concept general describe chapter 3 possible provide system call allow process create subprocesse execute concurrently aprocess need certain resource include cpu time memory file o device accomplish task resource typically allocate process run addition physical logical resource process obtain create initialization datum input pass example consider process run web browser function display content web page screen process give url input execute appropriate instruction system call obtain display desire information screen process terminate operate system reclaim emphasize program process program passive entity like content file store disk process active entity single threaded process program counter specify instruction execute thread cover chapter 4 exe- cution process sequential cpu execute instruction process process complete time instruction execute behalf process process associate program consider separate execution sequence multithreaded process multiple program counter point instruction execute give thread process unit work system system consist collec- tion process operate system process execute system code rest user process exe- cute user code process potentially execute concurrently multiplexe single cpu core parallel multiple cpu core operate system responsible following activity connec- tion process management create delete user

responsible following activity connec- tion process management create delete user system process schedule process thread cpu suspend resume process provide mechanism process synchronization provide mechanism process communication discuss process management technique chapter 3 chapter 7 discuss section 1.2.2 main memory central operation modern computer system main memory large array byte range size hundred thousand billion byte address main memory repository quickly accessible datum share cpu o device cpu read instruction main memory instruction fetch cycle read write datum main memory data fetch cycle von neumann architecture note early main memory generally large storage device cpu able address access directly example cpu process datum disk datum transfer main memory cpu generate o call way instruction memory cpu program execute map absolute address load memory program execute access program instruction datum memory generate absolute address eventually program terminate memory space declare available program load execute improve utilization cpu speed computer response user general purpose computer program memory create need memory management different memory- management scheme scheme reflect approach effectiveness give algorithm depend situation select memory management scheme specific system account factor especially hardware design system algorithm require hardware support operate system responsible following activity connec- tion memory management keep track part memory currently process allocate deallocate memory space need decide process part process datum memory memory management technique discuss chapter 9 chapter 10 computer system convenient user operate system provide uniform logical view information storage operate system abstract physical property storage device define logical storage unit fil operate system map file physical medium access file storage device file management visible component operate system computer store information different type physi- cal medium secondary storage common tertiary storage possible medium characteristic physical orga- nization control device disk drive unique characteristic property include access speed capacity data transfer rate access method sequential random file collection

capacity data transfer rate access method sequential random file collection related information define creator com- monly file represent program source object form datum data file numeric alphabetic alphanumeric binary file free- form example text file format rigidly example fix field mp3 music file clearly concept file extremely general operate system implement abstract concept file manag- e mass storage medium device control addition file normally organize directory easy use finally multiple user access file desirable control user access file user access example read write operate system responsible following activity connec- tion file management create delete file create delete directory organize file support primitive manipulate file directory map file mass storage back file stable nonvolatile storage medium file management technique discuss chapter 13 chapter 14 see computer system provide secondary storage main memory modern computer system use hdds nvm device principal line storage medium program datum program include compiler web browser word processor game store device load memory program use device source destination processing proper management secondary storage central importance computer system operate system responsible following activity connection secondary storage management mounting unmounting free space management storage allocation disk scheduling secondary storage frequently extensively efficiently entire speed operation computer hinge speed secondary storage subsystem algorithm manipulate time use storage slow low cost high capacity secondary storage backup disk datum storage seldom datum long term archival storage example magnetic tape drive tape cd dvd blu ray drive platter typical tertiary storage device tertiary storage crucial system performance manage operate system task leave tertiary storage management application program function operate system provide include mount unmounte medium device allocate free device exclusive use process migrate datum secondary tertiary storage technique secondary storage tertiary storage management discuss chapter 11 caching important principle computer system work information normally keep storage system main memory copy fast storage system

normally keep storage system main memory copy fast storage system cache tem- porary basis need particular piece information check cache use information directly cache access time ns < 1 kb < 1 tb < 10 tb disk tape characteristic type storage use information source put copy cache assumption need soon addition internal programmable register provide high speed cache main memory programmer compiler implement register- allocation register replacement algorithm decide information register main memory cache implement totally hardware instance system instruction cache hold instruction expect execute cache cpu wait cycle instruction fetch main memory similar reason system high speed datum cache memory hierarchy concern hardware cache text outside control operate system cache limited size cache management important design problem careful selection cache size replacement policy result greatly increase performance examine figure 1.14 replacement algorithm software control cache discuss movement information level storage hierarchy explicit implicit depend hardware design control- ling operating system software instance datum transfer cache cpu register usually hardware function operating system inter- vention contrast transfer datum disk memory usually control operate system hierarchical storage structure datum appear different level storage system example suppose integer incremente 1 locate file b file b reside hard disk increment operation proceed issue o operation copy disk block reside main memory operation follow copy cache internal register copy appear place hard disk main memory cache internal register figure 1.15 increment take place internal register value differ storage system value migration integer disk register new value write internal register hard disk compute environment process execute time arrangement pose difficulty access integer copy high level hierarchy multitaske environment cpu switch forth pro- cesse extreme care take ensure process wish access process obtain recently update value a. situation complicated multiprocessor environment addition maintain internal register cpu contain local cache refer figure 1.8 environment copy exist simultaneously

contain local cache refer figure 1.8 environment copy exist simultaneously cache cpu execute parallel sure update value cache immediately reflect cache reside situation call cache coherency usually hardware issue handle operate system level distribute environment situation complex environment copy replica file keep different computer replica access update concurrently distribute system ensure replica update place replica bring date soon possible way achieve guarantee discuss chapter 19 o system management purpose operate system hide peculiarity specific hardware device user example unix peculiarity o device hide bulk operate system o subsystem o subsystem consist component memory management component include buffering caching general device driver interface driver specific hardware device device driver know peculiarity specific device assign discuss early chapter interrupt handler device driver construction efficient o subsystem chapter 12 discuss o subsystem interface system component manage device transfer datum detect o completion security protection security protection computer system multiple user allow concurrent execution multiple process access datum regulate purpose mechanism ensure file memory segment cpu resource operate process gain proper authoriza- tion operate system example memory address hardware ensure process execute address space timer ensure process gain control cpu eventually relinquish control device control register accessible user integrity peripheral device protect protection mechanism control access process user resource define computer system mechanism provide mean specify control impose enforce control protection improve reliability detect latent error interface component subsystem early detection interface error prevent contamination healthy subsystem subsystem malfunction furthermore unprotected resource defend use misuse unauthorized incompetent user protection orient system provide means distinguish authorized unauthorized usage discuss chapter 17 system adequate protection prone failure allow inappropriate access consider user authentication information mean identify system steal datum copy delete file memory protection work job security defend system external internal attack attack spread huge range include virus worm denial of-service attack use system resource

include virus worm denial of- service attack use system resource legitimate user system identity theft theft service unauthorized use system prevention attack consider operating-system function system system leave policy additional software alarming rise security incident operating- system security feature fast grow area research implementa- tion discuss security chapter 16 protection security require system able distinguish user operate system maintain list user name asso- ciate user identifier user id windows parlance security id sid numerical id unique user user log system authentication stage determine appropriate user id user user id associate user process thread id need readable user translate user user list circumstance wish distinguish set user individual user example owner file unix system allow issue operation file select set user allow read file accomplish need define group set user belong group group functionality implement system wide list group name group identifier user group depend operating system design decision user group id include associate process course normal system use user id group id user sufficient user need escalate privilege gain extra permission activity user need access device restrict example operate system provide method allow privilege escalation unix instance setuid attribute program cause program run user id owner file current user id process run effective uid turn extra privilege terminate virtualization technology allow abstract hardware sin- gle computer cpu memory disk drive network interface card forth different execution environment create illu- sion separate environment run private computer environment view different individual operate system example windows unix run time interact user virtual machine switch operate system way user switch process run concurrently single operate system virtualization allow operating system run application operate system blush little reason func- tionality virtualization industry vast grow testa- ment utility importance broadly speak virtualization software member class include emulation emulation involve simulate computer hard- ware software typically

include emulation emulation involve simulate computer hard- ware software typically source cpu type different cpu intel x86 cpu desktop laptop computer include emulation facility call rosetta allow application compile entire operate system write platform run emula- tion come heavy price machine level instruction run natively source system translate equivalent function target system frequently result target instruction source target cpu similar performance level emulate code run slowly native code virtualization contrast operate system natively com- pile particular cpu architecture run operate system method multiple user run task concurrently run multiple vir- tual machine allow allow user run task system design single user later response problem run multiple microsoft windows application intel x86 cpu vmware create new virtualization technology form application run windows application run guest copy windows native x86 operate system run application figure 1.16 computer run single operating system b virtual windows host operating system vmware application virtual machine manager vmm vmm run guest operate system manage resource use protect guest modern operate system fully capable run multi- ple application reliably use virtualization continue grow laptop desktop vmm allow user install multiple operate system exploration run application write operate system native host example apple laptop run macos x86 cpu run windows 10 guest allow execution windows application com- panie write software multiple operate system use virtualization run operate system single physical server develop- ment testing debugging datum center virtualization common method execute manage compute environment vmm like vmware esxand citrix xenserver long run host operating system host operating system provide service resource management virtual machine process text provide linux virtual machine allow run linux development tool provide personal system regardless host operating system detail feature implementation virtualization find chapter 18 distributed system collection physically separate possibly heteroge- neous computer system network provide user access resource system maintain access shared resource

network provide user access resource system maintain access shared resource increase computation speed functionality datum availability reliability operating system generalize network access form file access detail networking contain network interface device driver user specifically invoke network function generally system contain mix mode example ftp nfs protocol create distribute system greatly affect system utility network simple term communication path system distributed system depend networking functional- ity network vary protocol distance node transport medium tcp ip common network protocol provide fundamental architecture internet operate system support tcp ip include general purpose one system support proprietary protocol suit need operate system necessary network protocol interface device network adapter example device driver manage software handle datum concept discuss book network characterize base distance node local area network lan connect computer room building campus wide area network wan usually link building city country aglobal company wan connect office worldwide example network run protocol protocol continue advent new technology bring new form network example metropolitan area network man link building city bluetooth 802.11 device use wireless technology communicate distance foot essence create personal area network pan phone headset smartphone desktop computer medium carry network equally varied include copper wire fiber strand wireless transmission satellite microwave dish radio compute device connect cellular phone create network short range infrared communication networking rudimentary level computer communicate use create network network vary performance operate system take concept network dis- tribute system notion provide network connectivity network operate system operating system provide feature file sharing network communication scheme allow different process different computer exchange message computer run network operate system act autonomously computer network aware network able communicate networked computer distribute operat- e system provide autonomous environment different computer communicate closely provide illusion single operat- e system control network cover computer network distribute system chapter 19 kernel data structures turn topic

network distribute system chapter 19 kernel data structures turn topic central operating system implementation way datum structure system section briefly describe fundamental datum structure extensively operating system reader kernel data structures singly link list require detail structure consult bibliography end chapter list stack queue array simple data structure element access directly example main memory construct array data item store large byte multiple byte allocate item item address item number × item size store item size vary remove item relative position remain item preserve situation array way data structure array list fundamental data structure com- puter science item array access directly item list access particular order list represent collec- tion datum value sequence common method implement structure link list item link linked list type singly link list item point successor illustrate doubly link list give item refer predecessor successor illustrate figure 1.18 circularly link list element list refer element null illustrate figure 1.19 linked list accommodate item vary size allow easy insertion deletion item potential disadvantage list per- formance retrieve specify item list size n linear o(n require potentially traverse n element bad case list some- time directly kernel algorithm frequently construct powerful data structure stack queue stack sequentially order data structure use lifo principle add remove item mean item doubly link list circularly link list place stack item remove operation insert remove item stack know push pop respectively operate system use stack invoke function call parameter local variable return address push stack function call return function pop item queue contrast sequentially order data structure use fifo principle item remove queue order insert everyday example queue include shopper wait checkout line store car wait line traffic signal queue common operate system job send printer typically print order submit example shall chapter 5 task wait run available cpu organize queue tree data

5 task wait run available cpu organize queue tree data structure represent datum hierarchically data value tree structure link parent child relationship general tree parent unlimited number child binary tree parent child term left child right child binary search tree additionally require ordering parent child left child < = right child figure 1.20 provide example binary search tree search item binary search tree bad case performance o(n consider occur remedy situation use algorithm create balanced binary search tree tree contain n item lg n level ensure bad case performance o(lg n shall section 5.7.1 linux use balanced binary search tree know red black tree cpu scheduling algorithm hash function map hash function take datum input perform numeric operation datum return numeric value numeric value index table typically array quickly retrieve datum search datum item list size n require o(n comparison hash function retrieve datum table good o(1 depend implementation detail performance hash function extensively operating system potential difficulty hash function unique input result output value link table kernel data structures binary search tree location accommodate hash collision have link list table location contain item hash value course collision efficient hash function use hash function implement hash map associate map key value pair hash function mapping estab- lishe apply hash function key obtain value hash map figure 1.21 example suppose user map password password authentication proceed follow user enter user password hash function apply user retrieve password retrieve password compare password enter user authentication abitmap string n binary digit represent status n item example suppose resource availability resource indicate value binary digit 0 mean resource available 1 indicate unavailable vice versa linux kernel data structure data structure linux kernel available kernel source code include file < linux list.h > provide detail link list datum structure kernel queue linux know kfifo

detail link list datum structure kernel queue linux know kfifo implementation find kfifo.c file kernel directory source code linux provide balanced binary search tree implementation red black tree detail find include file value ith position bitmap associate ith resource example consider bitmap show 0 0 1 0 1 1 1 0 1 resource 2 4 5 6 8 unavailable resource 0 1 3 7 available power bitmap apparent consider space efficiency use bit boolean value instead single bit result data structure time large bitmap commonly need represent availability large number resource disk drive provide nice illustration medium sized disk drive divide thousand individual unit call disk block bitmap indicate availability disk block summary data structure pervasive operating system implemen- tation structure discuss text explore kernel algorithm implementa- 1.10 computing environments far briefly describe aspect computer system operate system manage turn discussion operate system variety compute environment computing mature line

separate traditional com- put environment blur consider typical office environment year ago environment consist pc connect network server provide file print service remote access awkward portability achieve use laptop computer today web technology increase wan bandwidth stretch boundary traditional computing company establish portal pro- vide web accessibility internal server network computer thin clients)—which essentially terminal understand web base comput- ing place traditional workstation security easy maintenance desire mobile computer synchronize pc allow portable use company information mobile device connect wireless network cellular datum network use company web portal myriad web resource home user single computer slow modem con- nection office internet today network connection speed available great cost relatively inexpensive place give home user access datum fast data connection allow home computer serve web page run network include printer client pc server home use firewall pro- tect network security breach firewall limit communication device network half 20th century computing resource relatively scarce nonexistent period time system batch interactive batch system process job bulk prede- termine input file data source interactive system wait input user optimize use compute resource multiple user share time system time share system timer scheduling algorithm cycle process rapidly cpu give user share resource traditional time share system rare today scheduling tech- nique use desktop computer laptop server mobile computer frequently process own user single user operate system user process system process provide service user manage frequently get slice computer time consider window create user work pc example fact perform different task time web browser compose multiple process website currently visit time sharing apply web browser process mobile computing refer compute handheld smartphone tablet computer device share distinguish physical feature portable lightweight historically compare desktop laptop computer mobile system give screen size memory capacity overall functionality return handheld mobile access service e mail web browsing past year

handheld mobile access service e mail web browsing past year feature mobile device rich distinction functionality consumer laptop tablet computer difficult discern fact argue feature contemporary mobile device allow provide functionality unavailable impractical desktop today mobile system e mail web browsing play music video read digital book take photo record edit high definition video accordingly tremendous growth continue wide range application run device developer design application advantage unique feature mobile device global positioning system gps chip accelerometer gyroscope embedded gps chip allow mobile device use satellite determine precise location earth functionality especially useful design application provide navigation exam- ple tell user way walk drive direct nearby service restaurant accelerometer allow mobile device detect orientation respect ground detect certain force tilting shaking computer game employ accelerometer player interface system mouse keyboard tilt rotate shake mobile device practical use feature find augment reality appli- cation overlay information display current environment difficult imagine equivalent application develop traditional laptop desktop computer system provide access line service mobile device typically use ieee standard 802.11 wireless cellular datum network memory capacity processing speed mobile device limited pc smartphone tablet 256 gb storage uncommon find 8 tb storage desktop computer similarly power consumption concern mobile device use processor small slow offer few processing core processor find traditional desktop laptop computer operating system

currently dominate mobile computing apple ios google android ios design run apple iphone ipad mobile device android power smartphone tablet computer available manufacturer examine mobile operating system detail chapter 2 contemporary network architecture feature arrangement server system satisfy request generate client system form specialized distribute system call client server system general structure depict figure 1.22

server system broadly categorize compute server file compute server system provide interface client send request perform action example read datum response server execute action send result client server general structure client server system run database respond client request datum example system file serve system provide file system interface client create update read delete file example system web server deliver file client run web browser actual content file vary greatly range traditional web page rich multimedia content high definition video structure distribute system peer peer p2p system model model client server distinguish instead node system consider peer act client server depend request provide service peer peer system offer advantage traditional client server system client server system server bottleneck peer peer system service provide node distribute network participate peer peer system node join network peer node join network begin provide service request service node network determine service available accomplish general way node join network register service centralized lookup service network node desire specific service contact centralized lookup service determine node provide service remainder communication take place client service provider alternative scheme use centralized lookup service instead peer act client discover node provide desire service broadcast request service node network node node provide service respond peer make request support approach discovery protocol pro- vide allow peer discover service provide peer network figure 1.23 illustrate scenario peer peer network gain widespread popularity late 1990 file sharing service napster gnutella enable peer exchange file napster system approach simi- lar type describe centralized server maintain index file store peer node napster network actual exchange file take place peer node gnutella system tech- nique similar second type client broadcast file request node system node service request respond directly rial music example anonymously law govern reason future exchange file remain uncertain peer peer system centralized service skype example

file remain uncertain peer peer system centralized service skype example peer peer computing allow client voice call video call send text message internet technology know voice ip voip skype use hybrid peer- peer approach include centralized login server incorporate decentralized peer allow peer communicate cloud computing type computing deliver computing storage application service network way logical extension virtualization use virtualization base functionality example amazon elastic compute cloud ec2 facility thousand server million virtual machine petabyte storage available use internet user pay month base resource use actually type cloud computing include follow public cloud cloud available internet willing pay service private cloud cloud run company company use hybrid cloud cloud include public private cloud com- software service saas)—one application word processor spreadsheet available internet platform service paas)—a software stack ready application use internet example database server infrastructure service iaas)—server storage available internet example storage available make backup copy pro- cloud computing type discrete cloud computing environ- ment provide combination type example organiza- tion provide saas iaas publicly available service certainly traditional operating system type cloud infrastructure vmm manage virtual machine user process run high level vmm them- self manage cloud management tool vmware vcloud director open source eucalyptus toolset tool manage resource give cloud provide interface cloud component make good argument consider new type operating system figure 1.24 illustrate public cloud provide iaas.

notice cloud service cloud user interface protect firewall real time embedded systems embed computer prevalent form computer existence device find car engine manufacturing robot optical drive microwave oven tend specific task system run usually primitive operate system provide limited feature usually little user interface prefer spend time monitor manage hardware device automobile engine robotic arm embed system vary considerably general purpose computer run standard operate system linux special purpose application implement functionality hardware device special purpose embed operate system provide functionality desire hardware device application specific integrate circuit asic perform task operate system use embed system continue expand power device standalone unit element network web sure increase entire house computerize central computer general purpose computer embedded system control heating lighting alarm system coffee maker web access enable home owner tell house heat arrive home someday refrigerator able notify grocery store notice milk go embedded system run real time operate system areal- time system rigid time requirement place operation processor flow datum control device dedicated application sensor bring datum computer com- puter analyze datum possibly adjust control modify sensor input system control scientific experiment medical imaging system industrial control system certain display system real time system automobile engine fuel injection system home appliance controller weapon system real time system real time system define fix time constraint processing define constraint system fail instance robot arm instruct halt smash car build real time system function correctly return correct result time constraint contrast sys- tem traditional laptop system desirable mandatory respond quickly chapter 5 consider scheduling facility need implement real- time functionality operate system chapter 20 describe real time component linux free open source operate system study operating system easy avail- ability vast number free software open source release available source code code note free software open source software discussion topic free software refer

free software open source software discussion topic free software refer free libre software make source code available license allow cost use redistribution modification open source software necessarily offer licensing free software open source open source software free gnu linux famous open source operating system distribution free open source http://www.gnu.org/distros/ microsoft windows know example opposite closed source approach windows proprietary software microsoft own restrict use carefully protect source code

apple macos operate system comprise hybrid free open source operating systems approach contain open source kernel name darwin include proprietary closed source component start source code allow programmer produce binary neere source code binary lot work useful item comment recover learn operating system examine source code benefit source code hand student modify operate system compile run code try change excellent learning tool text include project involve modify operate system source code describe algorithm high level sure important operating system topic cover text provide pointer example open source code deep study benefit open source operating system include community interested usually unpaid programmer contribute code help write debug analyze provide support sug- gest change arguably open source code secure closed source code eye view code certainly open source code bug open source advocate argue bug tend find fix fast owe number people view code company earn revenue sell program hesitate show commercial company benefit suffer open source code revenue generate support contract sale hardware software run example early day modern computing 1950s software generally come source code

original hacker computer enthusiast mit tech model railroad club leave program drawer work homebrew user group exchange code meeting company- specific user group digital equipment corporation decus accept contribution source code program collect tape distribute tape interested member 1970 digital operate system computer software company eventually seek limit use software authorized computer pay customer release binary file compile source code source code help achieve goal protect code idea competitor homebrew user group 1970 exchange code meeting operate system hobbyist machine cpm proprietary 1980 proprietary software usual case free operating systems counter limit software use redistribution richard stallman 1984 start develop free unix compatible operate system call gnu(which recursive acronym gnu unix stallman free refer freedom use price free software movement object trade copy money hold user entitle certain freedom 1 freely run program 2 study change source code sell copy 3 4 change 1985 stallman publish gnu manifesto argue software free form free software foundation fsf goal encourage use development free software form licensing invent stallman copylefte work give possess copy work essential freedom work free condition redistribution preserve freedom gnu general public license gpl common license free software release fundamentally gpl require source code distribute binary copy include modify version release gpl license creative commons attribution sharealike license copyleft license sharealike way state idea copyleft example free open source operate system consider gnu linux 1991 gnu operate system nearly complete gnu project develop compiler editor utility library game part find gnu kernel ready prime time 1991 student finland linus torvalds release rudimentary unix like kernel gnu compiler tool invite contribution worldwide advent internet mean interested download source code modify submit change torvalds release update week allow call linux operate system grow rapidly

enhance thousand programmer 1991 linux free software license permit noncommercial redistribution 1992 torvalds rerelease linux gpl make free software use term coin later open source resulting gnu linux operate system kernel properly call linux operating system include gnu tool call gnu linux spawn hundred unique distribution custom build system major distribution include red hat suse fedora debian slackware ubuntu distribution vary function utility instal application hardware support user interface purpose example red hat enterprise linux gear large commercial use pclinuxos live cd operate system boot run cd rom instal system boot disk variant pclinuxos call pclinuxos supergamer dvd live dvd include graphic driver game gamer run compatible system simply boot dvd gamer finish reboot system reset instal operating system run linux windows system following simple free approach free open source operating system 1 download free virtualbox vmm tool install system 2 choose install operate system scratch base installation image like cd choose pre built operating system image instal run quickly site like image preinstalle operate system application include flavor gnu linux 3 boot virtual machine virtualbox alternative virtualbox use free program qemu http://wiki.qemu.org/download/ include qemu img command convert virtualbox image qemu image easily import text provide virtual machine image gnu linux run ubuntu release image contain gnu linux source code tool software development cover example involve gnu linux image text detailed case study chapter 20 bsd unix long complicated history linux start 1978 derivative at&t unix release university california berkeley ucb come source binary form open source license at&t require bsd unix development slow lawsuit at&t eventually fully functional open source version 4.4bsd lite release 1994 linux distribution bsd unix include freebsd netbsd openbsd dragonflybsd explore source code freebsd simply download virtual machine image version interest boot virtualbox describe linux source code come distribution store /usr src/.

kernel source code /usr src sys example examine vir- tual memory implementation code freebsd kernel file /usr src sys vm alternatively simply view source code online open source project source code contain control version control system case subversion https://subversion.apache.org/source-code version control system allow user pull entire source code tree computer push change repository pull system provide feature include entire history file conflict resolution feature case file change concurrently version control system git gnu linux darwin core kernel component macos base unix open source source code available http://www.opensource.apple.com/. macos release open source component post site package contain kernel begin xnu apple provide extensive developer tool documentation support http://developer.apple.com study operate system interesting time study operating system easy open source movement overtake oper- ating system cause available source binary executable format list operating system available format include linux bsd unix solaris macos availabil- ity source code allow study operating system inside question answer look documentation behavior operate system answer examine operate system long commercially viable open source enable study system operate time few cpu memory storage resource extensive incomplete list open source operating system project available http://dmoz.org/computers/software/operating systems open source/.

addition rise virtualization mainstream frequently free computer function make possible run operate system core system example vmware http://www.vmware.com pro- vide free player windows hundred free virtual appli- ances run virtualbox http://www.virtualbox.com provide free open source virtual machine manager operate system tool student try hundred operate system ded- case simulator specific hardware available allow- e operate system run native hardware con- fine modern computer modern operate system example decsystem-20 simulator run macos boot tops-20 load source tape modify compile new tops-20 kernel interested student search internet find original paper describe operate system original manual advent open source operating system easy student operating system developer knowledge effort internet connection student create new operating system distribution year ago difficult impossible access source code access limit interest time disk space student solaris commercial unix base operating system sun microsystems originally sun sunos operate system base bsd unix sun move at&t system v unix base 1991 2005 sun open source solaris code opensolaris project purchase sun oracle 2009 leave state project unclear group interest opensolaris expand feature work set project illumos expand open- solaris base include feature basis product illumos available http://wiki.illumos.org open source system learning tool free software movement drive legion programmer create thousand open source project include operating system site like http://freshmeat.net/ http://distrowatch.com/

provide portal project state early open source project enable student use source code learning tool modify program test help find fix bug explore mature featured operating system compiler tool user interface type program availability source code historic project multics help stu- dent understand project build knowledge help implementation new project advantage work open source operating system diversity gnu linux bsd unix open source operating system instance goal utility licensing purpose some- time license mutually exclusive cross pollination occur allow- e rapid improvement operate system project example major component opensolaris port bsd unix advan- tage free software open sourcing likely increase number quality open source project lead increase number individual company use project operate system software manage computer hardware provide environment application program run interrupt key way hardware interact operate system hardware device trigger interrupt send signal cpu alert cpu event require attention interrupt manage interrupt handler computer job execute program program main memory large storage area processor access directly main memory usually volatile storage device lose content power turn lose nonvolatile storage extension main memory capable hold large quantity datum permanently common nonvolatile storage device hard disk provide storage program datum wide variety storage system computer system organize hierarchy accord speed cost high level expensive fast hierarchy cost bit generally decrease access time generally increase modern computer architecture multiprocessor system cpu contain compute core well utilize cpu modern operate system employ multiprogram- ming allow job memory time ensure cpu job execute multitasking extension multiprogramming cpu schedul- e algorithm rapidly switch process provide user fast response time prevent user program interfere proper operation system system hardware mode user mode kernel instruction privileged execute kernel mode example include instruction switch kernel mode o control timer management interrupt management process fundamental unit work operate system pro- cess management include create delete process provide

operate system pro- cess management include create delete process provide mechanism process communicate synchronize operate system manage memory keep track part memory responsible dynami- cally allocate free memory space storage space manage operate system include provide file system represent file directory manage space operate system provide mechanism protect secure operate system user protection measure control access process user resource available computer system virtualization involve abstract computer hardware different execution environment data structure operate system include list stack queue tree map computing take place variety environment include traditional computing mobile computing client server system peer peer sys- tem cloud computing real time embed system free open source operate system available source code for- mat free software license allow cost use redistribution modification gnu linux freebsd solaris example popular main purpose operate system stress need operate system efficient use compute hardware appropriate operate system forsake principle waste resource system wasteful main difficulty programmer overcome write operate system real time environment keep mind definition operate system consider operate system include application web browser mail program argue support answer distinction kernel mode user mode function rudimentary form protection security follow instruction privileged set value timer read clock issue trap instruction turn interrupt modify entry device status table switch user kernel mode access o device early computer protect operate system place memory partition modify user job operate system describe difficulty think arise scheme cpu provide mode operation possible use multiple mode timer compute current time provide short description accomplish reason cache useful problem solve problem cause cache large device cache instance cache large disk large eliminate device distinguish client server peer peer model dis- general textbook cover operating system include stallings 2017 tanenbaum 2014 hennessy patterson 2012 provide coverage o system bus system architecture general kurose ross 2017 provide general overview computer network

russinovich et al 2017 overview microsoft windows cov- er considerable technical detail system internal component mcdougall mauro 2007 cover internal solaris operate system macos ios internal discuss levin 2013 levin 2015 cover internal android love 2010 provide overview linux operate system great detail data structure linux kernel free software foundation publish philosophy hennessy patterson 2012 j. hennessy d. patterson computer archi- tecture quantitative approach fifth edition morgan kaufmann 2012 kurose ross 2017 j. kurose k. ross computer networking approach seventh edition addison wesley 2017 j. levin mac os x ios internals apple core wiley j. levin android internals confectioner cookbook volume r. love linux kernel development edition developer mcdougall mauro 2007 r. mcdougall j. mauro solaris internals second edition prentice hall 2007 russinovich et al 2017 m. russinovich d. a. solomon a. ionescu win- dows internals 1

seventh edition microsoft press 2017 w. stallings operating systems internals design principles 9th edition ninth edition prentice hall 2017 a. s. tanenbaum modern operating systems prentice hall chapter 1 exercise clustered system differ multiprocessor system require machine belong cluster cooperate provide highly available service consider compute cluster consist node run database describe way cluster software manage access datum disk discuss benefit disadvantage purpose interrupt interrupt differ trap trap generate intentionally user program explain linux kernel variable hz jiffy determine number second system run direct memory access high speed o device order avoid increase cpu execution load cpu interface device coordinate cpu know memory operation com- cpu allow execute program dma controller transfer datum process interfere execution user program describe form interference cause computer system provide privileged mode operation hardware possible construct secure operate system computer system argument smp system different level cache level local processing core level share processing core cache system design way rank following storage system slow fast consider smp system similar show figure 1.8 illustrate example datum reside memory fact different value local cache discuss example problem maintain coherence cache datum manifest following processing environment describe mechanism enforce memory protection order prevent program modify memory associate network configuration lan wan well suit fol- campus student union campus location statewide university system describe challenge design operate system mobile device compare design operate system tradi- advantage peer peer system client server describe distribute application appropriate identify advantage disadvantage open source operating system identify type people find aspect advantage disadvantage c h p t e r operating system provide environment program execute internally operating system vary greatly makeup organize different line design new operating system major task important goal system define design begin goal form basis choice algorithm strategy view operate system vantage point view focus service system provide interface

operate system vantage point view focus service system provide interface make available user programmer component interconnection chapter explore aspect operating system show viewpoint user programmer operating system designer consider service operate system provide provide debug methodology design system finally describe operate system create computer start operate system identify service provide operate system illustrate system call provide operating system service compare contrast monolithic layered microkernel modular hybrid strategy design operate system illustrate process boot operate system apply tool monitor operate system performance design implement kernel module interact linux kernel operate system provide environment execution program make certain service available program user pro- grams specific service provide course differ operate user system program view operating system service system identify common class figure 2.1 show view operating system service interrelate note service programming task easy programmer set operating system service provide function helpful user interface operate system user interface ui interface form commonly graphical user interface gui interface window system mouse serve point device direct o choose menu selection keyboard enter text mobile system phone tablet provide touch screen interface enable user slide finger screen press button screen select choice option command line interface cli use text command method enter keyboard type command specific format specific option system provide variation program execution system able load program mem- ory run program program able end execu- tion normally abnormally indicate error o operation run program require o involve file o device specific device special function desire read network interface write file system efficiency protection user usually control o device directly operate system provide means o. file system manipulation file system particular interest obvi- ously program need read write file directory need create delete search give file list file infor- mation finally operate system include permission management allow deny access file directory base file ownership operating

management allow deny access file directory base file ownership operating system provide variety file system allow personal choice provide specific feature performance communication circumstance process need exchange information process communication occur process execute computer process execute different computer system tie network communication implement shared memory process read write share section memory message passing packet information predefined format move process operate system error detection operate system need detect correct error constantly error occur cpu memory hardware memory error power failure o device parity error disk connection failure network lack paper printer user program arithmetic overflow attempt access illegal memory location type error operate system appropriate action ensure correct consistent computing choice halt system time terminate error cause process return error code process process detect possibly correct set operate system function exist help user ensure efficient operation system system multiple process gain efficiency share computer resource different process resource allocation multiple process run time resource allocate operate system manage different type resource cpu cycle main memory file storage special allocation code o device general request release code instance determine well use cpu operate system cpu scheduling routine account speed cpu process execute number processing core cpu factor routine allocate printer usb storage drive peripheral device logging want track program use kind computer resource record keeping accounting user bill simply accumulate usage statistic usage statistic valuable tool system administrator wish reconfigure system improve compute service protection security owner information store multiuser networked computer system want control use informa- tion separate process execute concurrently possible process interfere oper- ate system protection involve ensure access system resource control security system outsider impor- tant security start require user authenticate system usually mean password gain access system resource extend defend external o device includ- e network adapter invalid access attempt record connection detection break in system protect

invalid access attempt record connection detection break in system protect secure precaution institute chain strong weak link user operating system interface mention early way user interface operate system discuss fundamental approach provide command line interface command interpreter allow user directly enter command perform operate system allow user interface operate system graphical user interface gui operate system include linux unix windows treat com- mand interpreter special program run process ini- tiate user log interactive system system multiple command interpreter choose interpreter know shell example unix linux system user choose sev- eral different shell include c shell bourne shell korn shell party shell free user write shell available shell provide similar functionality user choice shell use generally base personal preference figure 2.2 show bourne bash shell command interpreter macos main function command interpreter execute user specify command command give level manipu- late file create delete list print copy execute shell available unix system operate way command imple- mente general way approach command interpreter contain code exe- cute command example command delete file cause command interpreter jump section code set parameter make appropriate system case number command give determine size command interpreter command require implement code alternative approach unix operating system implement command system program case command interpreter understand command way merely use command identify file load memory execute unix command delete file search file call rm load file memory execute parameter file.txt logic associate rm command user operating system interface bash shell command interpreter macos define completely code file rm way programmer add new command system easily create new file proper program logic command interpreter program small change new command add graphical user interface second strategy interface operate system user- friendly graphical user interface gui enter command directly command line interface user employ mouse base window- menu system characterize desktop metaphor user

employ mouse base window- menu system characterize desktop metaphor user move mouse position pointer image icon screen desktop represent program file directory system function depend mouse pointer location click button mouse invoke program select file directory know folder pull menu contain command graphical user interface appear research take place early 1970s xerox parc research facility gui appear xerox alto computer 1973 graphical interface widespread advent apple macintosh computer 1980 user interface macintosh operate system undergo change year significant adoption aqua interface appear macos microsoft version windows version 1.0 base addition gui interface ms dos operate system later version windows significant change appearance gui enhancement functionality traditionally unix system dominate command line inter- face gui interface available significant develop- ment gui design open source project k desktop environment kde gnome desktop gnu project kde gnome desktop run linux unix system available open source license mean source code readily available reading modification specific license term command line interface mouse keyboard system impractical mobile system smartphone handheld tablet com- puter typically use touch screen interface user interact make gesture touch screen example press swipe finger screen early smartphone include physical keyboard smartphone tablet simulate keyboard touch screen figure 2.3 illustrate touch screen apple iphone ipad iphone use springboard touch screen interface choice interface choice use command line gui interface personal preference system administrator manage computer power user deep knowledge system frequently use iphone touch screen user operating system interface command line interface efficient give fast access activity need perform system subset system function available gui leave common task command line knowledgeable command line inter- face usually repetitive task easy programmability example frequent task require set command- line step step record file file run like program program compile executable code interpret command line interface shell script common system command line orient unix contrast windows

shell script common system command line orient unix contrast windows user happy use windows gui envi- ronment use shell interface recent version win- dows operate system provide standard gui desktop tradi- tional laptop touch screen tablet change undergo macintosh operate system provide nice study contrast his-

torically mac os provide command line interface require user interface operate system gui release macos implement unix kernel oper- ating system provide aqua gui command line interface figure 2.4 screenshot macos gui app provide command line interface ios android mobile system rarely instead user mobile system interact device touch screen interface user interface vary system system user user system typically substantially remove actual system structure design useful intuitive user interface direct function operate system book concentrate fundamental problem provide adequate service macos gui user program point view operate system distinguish user program system program system call provide interface service available operat- e system call generally available function write c c++ certain low level task example task hardware access directly write assembly language discuss operate system make system call available let use example illustrate system call write simple program read datum file copy file input program need name file input file output file name specify way depend operate system design approach pass name file command example unix cp command cp in.txt out.txt command copy input file in.txt output file out.txt

sec- ond approach program ask user name interactive system approach require sequence system call write prompt message screen read keyboard character define file mouse base icon base system menu file name usually display window user use mouse select source window open destination specify sequence require o system call file name obtain program open input file create open output file operation require system possible error condition system handle example program try open input file find file file protect access case program output error message sequence system call terminate abnormally system input file exist create new output file find output file situation cause program abort system delete exist file system create new system option interactive system ask user sequence system call output prompt message read response terminal replace exist file abort program file set enter loop read input file system write output file system read write return status information possible error condition input program find end file example system sequence acquire input file write prompt screen acquire output file write prompt screen open input file file exist abort create output file file exist abort read input file write output file read fail close output file write completion message screen example system call reach hardware failure read parity error write operation encounter error depend output device example available disk space finally entire file copy program close file system call write message console window system call finally terminate normally final system system sequence show figure 2.5 application programming interface simple program heavy use operat- e system frequently system execute thousand system call second programmer level detail typically applica- tion developer design program accord application programming interface api api specify set function available appli- cation programmer include parameter pass function return value programmer expect common api available application programmer windows api windows system posix api

api available application programmer windows api windows system posix api posix base system include virtually version unix linux macos java api program run java virtual machine aprogrammer access api library code provide operate system case unix linux program write c language library call libc note specify system name text generic example operate system system scene function api typically invoke actual system call behalf application programmer example windows function createprocess unsurprisingly create example standard api example standard api consider read function avail- able unix linux system api function obtain man page invoke command command line description api appear read(int fd void buf size_t count aprogram use read function include unistd.h header file file define ssize t size t datum type thing parameter pass read follow int fd file descriptor read void buf buffer datum read size t count maximum number byte read successful read number byte read return return value 0 indicate end file error occur read return 1 new process actually invoke ntcreateprocess system application programmer prefer program accord api invoke actual system call reason mer design program api expect program compile run system support api reality architectural difference difficult appear furthermore actual system call detailed difficult work api available application programmer exist strong correlation function api associate system kernel fact posix windows api similar native system call provide unix linux windows operating important factor handle system call run time envi- ronment rte)—the suite software need execute application writ- give programming language include compiler interpreter software library loader rte provide open system interface handling user application invoke open system system interface serve link system call available operate system system interface intercept function call api invoke necessary system call operate system typically number associate system system interface maintain table index accord number system- interface invoke intend system operate system kernel return

number system- interface invoke intend system operate system kernel return status system caller need know system implement execution caller need obey api understand operate system result execution system detail operate system interface hide programmer api manage rte relationship api system interface operate system show figure 2.6 illustrate operate system handle user application invoke open system system call occur different way depend computer use information require simply identity desire system exact type information vary accord particular operate system example input need specify file device use source address length memory buffer input read course device file length implicit general method pass parameter operate sys- tem simple approach pass parameter register case parameter register case parameter generally store block table memory address block pass parameter register figure 2.7 linux use combination approach few parameter table x load address x system 13 passing parameter table register parameter block method parameter place push stack program pop stack operate system operate system prefer block stack method approach limit number length parameter pass type system calls system call group roughly major category process control fil management device management information maintenance communi- cation protection briefly discuss type system call provide operate system system call support support concept function discuss late chap- ter figure 2.8 summarize type system call normally provide operate system mention text normally refer system call generic name text provide example actual counterpart system call unix linux windows run program need able halt execution normally end abnormally abort system terminate currently run program abnormally program run problem cause error trap dump memory take error message generate dump write special log file disk examine debugger system program design aid programmer find correct error bug determine cause problem normal abnormal circumstance operate system transfer control invoke command interpreter command interpreter read command interactive system command interpreter simply continue

command interpreter read command interactive system command interpreter simply continue command assume user issue appropriate command respond process control create process terminate process process attribute set process attribute wait event signal event allocate free memory file management create file delete file read write reposition file attribute set file attribute device management request device release device read write reposition device attribute set device attribute logically attach detach device information maintenance time date set time date system datum set system datum process file device attribute set process file device attribute create delete communication connection send receive message transfer status information attach detach remote device file permission set file permission type system call example windows unix system call follow illustrate equivalent system call windows unix operate system error gui system pop window alert user error ask guidance system allow special recovery action case error occur program discover error input want terminate abnormally want define error level severe error indicate high level error parameter possible combine normal abnormal termination define normal termination error level 0 command interpreter follow program use error level determine action automatically process execute program want load execute program feature allow command interpreter execute program direct example user command click mouse interesting question return control load program terminate question relate existing program lose save allow continue execution concurrently new program control return exist program new program termi- nate save memory image exist program standard c library standard c library provide portion system interface version unix linux example let assume c pro- gram invoke printf statement c library intercept invoke necessary system call operate system instance write system c library take value return write pass user program int main standard c

value return write pass user program int main standard c library effectively create mechanism program program program continue concurrently create new process multiprogramme system specifically purpose create new process set process able control execution control require ability determine reset attribute process include process priority max- imum allowable execution time process attribute set process attribute want terminate process create terminate process find incorrect long having create new process need wait finish execution want wait certain time pass wait time probably want wait specific event occur wait event process signal event occur signal event process share datum ensure integrity datum share operate system provide system call allow arduino execution system startup b run sketch process lock share datum process access datum lock release typically system call include acquire lock release lock system call type deal coordination concurrent process discuss great detail chapter 6 chapter 7 facet variation process control use example involve single task system multitaske system clarify concept arduino simple hardware platform consist microcontroller input sensor respond variety event change light temperature barometric pressure write program arduino write program pc upload compile program know sketch pc arduino flash memory usb connection standard arduino platform provide operate system instead small piece software know boot loader load sketch specific region arduino memory figure 2.9 sketch load begin run wait event program respond example arduino temperature sensor detect temperature exceed certain threshold sketch arduino start motor fan arduino consider single task system sketch present memory time sketch load replace exist sketch furthermore arduino provide user interface hardware input sensor freebsd derive berkeley unix example multitaske system user log system shell user choice run await command run program user request freebsd multitaske system command interpreter continue run program execute figure 2.10 start new pro- cess shell execute fork system select program load memory exec system

shell execute fork system select program load memory exec system program execute depend command issue shell wait process finish run process background case shell immediately wait command enter process run background receive input directly keyboard shell resource o file gui interface user free ask shell run program monitor progress run process change program priority process execute exit freebsd run multiple program system terminate return invoke process status code 0 nonzero error code status error code available shell program process discuss chapter 3 program example fork exec system call file system discuss detail chapter 13 chapter 15 identify common system call deal file need able create delete file system require file file attribute file create need open use read write reposition rewind skip end file example finally need close file indicate long need set operation directory directory structure organize file file system addition file directory need able determine value attribute set necessary file attribute include file file type protection code accounting information system call file attribute set file attribute require function operate system provide call call file copy provide api perform operation code system call provide system program perform task system program callable program consider api system process need resource execute main memory disk drive access file resource available grant control return user process process wait sufficient resource available resource control operate system think device device physical device example disk drive think abstract virtual device example file asystem multiple user require request device ensure exclusive use finish device release function similar open close system call file operate system allow unmanaged access device hazard potential device contention deadlock describe chapter 8 device request allocate read write possibly reposition device file fact similarity o device file great operate system include unix merge combine file device structure case set system call file device o

file device structure case set system call file device o device identify special file name directory placement file user interface file device appear similar underlie system call dissimilar example design decision build operate system user system call exist simply purpose transfer information user program operate system example sys- tem system return current time date system call return information system version number operate system free memory disk space set system call helpful debug program system provide system call dump memory provision useful debugging program strace available linux system list system execute microprocessor provide cpu mode know single step trap execute cpu instruction trap usually catch debugger operate system provide time profile program indicate time program execute particular location set location time profile require tracing facility regular timer interrupt occurrence timer interrupt value program counter record sufficiently frequent timer interrupt statistical picture time spend part program obtain addition operate system keep information process system call access information generally call set process information process attribute set process attribute section 3.1.3 discuss information common model interprocess communication message- pass model share memory model message passing model communicate process exchange message trans- fer information message exchange process directly indirectly common mailbox communication place connection open communica- tor know process system process computer connect communication network computer network host commonly know host network identifier ip address similarly process process translate identifier operate system refer process hostid processid system call translation identifier pass general- purpose open close call provide file system specific open connection close connection system call depend system model communication recipient process usually permission communication place accept connection process receive connection special purpose dae- mons system program provide purpose execute wait connection awaken connection source communication know client receive dae- mon know server exchange message read message write message system call close connection terminate share

read message write message system call close connection terminate share memory model process use share memory create share memory attach system call create gain access region memory own process recall normally operate system try prevent process access process memory share memory require process agree remove restriction exchange information read write datum share area form datum determine process operate system control process responsible ensure write location simultaneously mechanism discuss chapter 6 chapter 4 look variation process scheme thread memory share default model discuss common operate system system implement message passing useful exchange small amount datum conflict need avoid eas- i implement share memory intercomputer communication share memory allow maximum speed convenience communication memory transfer speed take place computer problem exist area protection synchroniza- tion process share memory protection provide mechanism control access resource pro- vide computer system historically protection concern multiprogrammed computer system user advent networking internet computer system server mobile handheld device concern protection typically system call provide protection include set permission permission manipulate permission setting resource file disk allow user deny user system call specify particular user allow access certain resource cover protection chapter 17 large issue security involve protection external threat chapter 16 aspect modern system collection system service recall figure 1.1 depict logical computer hierarchy low level hardware operate system system service finally application program system service know system utility provide convenient environment program development execution simply user interface system call consider- ably complex divide category file management program create delete copy rename print list generally access manipulate file directory status information program simply ask system date time available memory disk space number user similar status information complex provide detailed performance logging debugging information typically pro- grams format print output terminal output device file display window gui system support registry store retrieve configuration information file modificatio text

system support registry store retrieve configuration information file modificatio text editor available create mod- ify content file store disk storage device special command search content file perform transfor- mation text programming language support compiler assembler debugger interpreter common programming language c c++ java python provide operate system available separate download program loading execution program assemble com- pile load memory execute system provide absolute loader relocatable loader linkage editor overlay loader debugging system high level language machine language need communication program provide mechanism create virtual connection process user computer system allow user send message screen browse web page send e mail message log remotely transfer file machine background service general purpose system method launch certain system program process boot time process terminate complete task continue linkers loaders run system halt constantly run system program pro- cesse know service subsystem daemon example network daemon discuss section 2.3.3.5 example sys- tem need service listen network connection order connect request correct process example include process scheduler start process accord specified schedule system error monitor service print server typical system dozen daemon addition operate system run important activity user context kernel context use daemon run system program operating system supply program useful solve common problem perform common operation application program include web browser word proces- sor text formatter spreadsheet database system compiler plotting statistical analysis package game view operate system see user define application system program actual system call con- sider user pc user computer run macos operating system user gui feature mouse window interface alternatively window user command- line unix shell use set system call system call look different act different way confuse user view consider user dual boot macos windows user hardware entirely different interface set applica- tion physical resource hardware user expose multiple user interface sequentially concurrently linkers loaders usually program reside

multiple user interface sequentially concurrently linkers loaders usually program reside disk binary executable file example a.out prog.exe run cpu program bring mem- ory place context process section describe step procedure compile program place memory eligible run available cpu core step highlight source file compile object file design load physical memory location format know relocatable object fil linker combine relocatable object file single binary executable file linking phase object file library include standard c math library specify flag loader load binary executable file memory eligible run cpu core activity associate linking loading relocation assign final address program part adjust code datum program match address example code library function access variable execute figure 2.11 run loader necessary enter executable file command line program enter gcc -c main.c gcc -o main main.o -lm role linker loader command line unix system example ./main shell create new process run program fork system shell invoke loader exec system pass exec executable file loader load specify program memory address space newly create process gui interface double click icon associate executable file invoke loader similar mechanism process describe far assume library link executable file load memory reality system allow program dynamically link library program load windows instance support dynamically link library dlls benefit approach avoid link load library end executable file instead library conditionally link load require program run time example figure 2.11 math library link executable file main linker insert relocation information allow dynamically link load program load shall chapter 9 possible multiple process share dynamically link library result significant saving memory use object file executable file typically standard format include compile machine code symbol table contain metadata function variable reference program unix linux system standard format know elf executable linkable format separate elf format relocatable application operating system specifi linux provide command identify evaluate elf file

operating system specifi linux provide command identify evaluate elf file example file command determine file type main.o object file main executable file command report main.o elf relocatable file command report main elf executable elf file divide number section evaluate readelf command executable file piece information elf file executable file program entry point contain address instruction execute program run windows system use portable executable pe format macos use mach o format application operate system specific fundamentally application compile operate system exe- cutable operate system world well place choice operating system use depend utility feature application available base early discussion problem operate system provide unique set system call system call set service provide operate system use application system call uniform barrier difficult execute application program different operate system multiple operate system application possible application available run multiple operate system way 1 application write interpret language python ruby interpreter available multiple operate system interpreter read line source program execute equivalent instruction native instruction set call native operate sys- tem call performance suffer relative native application interpreter provide subset operate system feature possibly limit feature set associate application 2 application write language include virtual machine contain run application virtual machine language rte example method java java rte include loader byte code verifier component load java application java virtual machine rte port develop operate system mainframe smartphone theory java app run rte available system kind disadvantage similar interpreter discuss 3 application developer use standard language api compiler generate binary machine- operating system- specific language application port operate sys- tem run porting time consume new version application subsequent testing debugging well know example posix api set standard maintain source code compatibility different variant unix like operating system theory approach seemingly provide simple solution develop application run different operating system how- general lack application mobility cause develop

different operating system how- general lack application mobility cause develop cross platform application challenging task application level library provide operate system contain api provide feature like gui interface application design set api available ios apple iphone work operate system provide api android challenge exist low level system include follow operate system binary format application dictate layout header instruction variable component need certain location specify structure executable file operate system open file load application cpu varying instruction set application contain appropriate instruction execute correctly operate system provide system call allow application request activity create file open network connec- tion system call vary operate system respect include specific operand operand ordering appli- cation invoke system call numbering number mean- ing return result approach helped address com- pletely solve architectural difference example linux unix system adopt elf format binary executable file elf provide common standard linux unix system elf format tie specific computer architecture guarantee executable file run different hardware platform api mention specify certain function application level architecture level application binary interface abi define different component binary code interface given operating system given architecture abi specify low level detail include address width method pass parameter system call organization operating system design implementation run time stack binary format system library size data type typically abi specify given architecture example abi armv8 processor abi architecture level equivalent api binary executable file compile link accord particular abi able run different system support abi particular abi define certain operating system run give architecture abi little provide cross platform compatibility sum difference mean interpreter rte binary executable file write compile specific operate system specific cpu type intel x86 armv8 application fail run imagine work require program firefox browser run windows macos linux release ios android cpu architecture operating system design implementation section discuss problem face design implement

operating system design implementation section discuss problem face design implement operate system course complete solution problem approach prove successful problem design system define goal specification high level design system affect choice hard- ware type system traditional desktop laptop mobile distributed real time high design level requirement hard specify requirement divide basic group user goal system goal user want certain obvious property system system convenient use easy learn use reliable safe fast course specification particularly useful system design general agreement achieve similar set requirement define developer design create maintain operate system system easy design implement maintain flexible reliable error free efficient requirement vague interpret short unique solution problem define require- ment operate system wide range system existence show different requirement result large variety solution different environment example requirement wind river vxworks real- time operate system embed system substantially different windows server large multiaccess operate system design enterprise application specify design operate system highly creative task textbook tell general principle develop field software engineering turn discus- sion principle mechanism policies important principle separation policy mechanism mecha- nism determine policy determine example timer construct section 1.4.3 mechanism ensure cpu protection decide long timer set particular user policy decision separation policy mechanism important flexibility policy likely change place time bad case change policy require change underlie mechanism general mechanism flexible work range policy preferable change policy require redefinition certain parameter system instance consider mechanism give priority certain type program mechanism properly separate policy support policy decision o intensive program priority cpu intensive one support microkernel base operate system discuss section 2.8.3 separation mechanism policy extreme implement basic set primitive building block block policy free allow advanced mechanism policy add user create kernel module user program contrast consider windows enormously popular commercial operating system available decade microsoft closely encode mechanism policy system enforce global look

decade microsoft closely encode mechanism policy system enforce global look feel device run windows operate system application similar interface interface build kernel system library apple adopt similar strategy macos ios operate system similar comparison commercial open source operate system instance contrast windows discuss linux open source operate system run wide range com- put device available 25 year standard linux kernel specific cpu scheduling algorithm cover section 5.7.1 mechanism support certain policy free modify replace scheduler support different policy policy decision important resource allocation necessary decide allocate resource policy decision question mechanism determine operate system design implement oper- ating system collection program write people long period time difficult general statement early operating system write assembly language write high level language c c++ small amount system write assembly language fact higher- level language low level kernel write assembly language c. high level routine write c c++ system library write c++ high level lan- guage android provide nice example kernel write c assembly language android system library write c c++ application framework provide developer interface system write java cover android architecture detail section 2.8.5.2 advantage high level language systems-implementation language implement operate system gain language application program code write fast compact easy understand debug addition improvement compiler technology improve gener- ate code entire operate system simple recompilation finally operate system far easy port hardware write high level language particularly important operating system intend run different hardware system small embed device intel x86 system arm chip run phone possible disadvantage implement operate system high level language reduced speed increase storage requirement major issue today system expert assembly language programmer produce efficient small routine large program modern compiler perform complex analysis apply sophis- ticate optimization produce excellent code modern processor deep pipelining multiple functional unit handle detail complex dependency easily human mind true system

unit handle detail complex dependency easily human mind true system major performance improvement operate system likely result well datum structure algorithm excellent assembly language code addition operate sys- tem large small code critical high performance interrupt handler o manager memory manager cpu scheduler probably critical routine system write work correctly bottleneck identify refactore operate system large complex modern operate system engi- neere carefully function properly modify easily common approach partition task small component module single system module define portion system carefully define interface function use similar approach structure program place code main function instead separate logic num- ber function clearly articulate parameter return value function main shell command compiler interpreter system interface kernel character o system swapping block o disk tape driver kernel interface hardware disk tape traditional unix system structure briefly discuss common component operate system chapter 1 section discuss component interconnect meld kernel simple structure organize operate system structure place functionality kernel single static binary file run single address space approach know monolithic structure common technique design operate system example limited structuring original unix operating system consist separable part kernel system program kernel separate series interface device driver add expand year unix evolve view traditional unix operate system layer extent show figure 2.12 system interface physical hardware kernel kernel provide file system cpu scheduling memory management operating- system function system call take sum enormous functionality combine single address space linux operate system base unix structure similarly show figure 2.13 application typically use glibc standard c library communicate system interface kernel linux kernel monolithic run entirely kernel mode single address space shall section 2.8.4 modular design allow kernel modify run time despite apparent simplicity monolithic kernel difficult implement extend monolithic kernel distinct performance advantage little overhead system interface communication kernel fast despite drawback glibc standard c library linux system

kernel fast despite drawback glibc standard c library linux system structure monolithic kernel speed efficiency explain evidence structure unix linux windows operate system monolithic approach know tightly couple system change system wide range effect part alternatively design loosely couple system system divide separate small component specific limited functionality component comprise kernel advantage modular approach change component affect component allow system implementer freedom create change inner working system system modular way method layered approach operate system break number layer level layer layer 0 hardware high layer n user interface layer structure depict figure 2.14 operate system layer implementation abstract object datum operation manipulate datum typical operate system layer layer m consist datum structure set function invoke high level layer layer m turn invoke operation low level layer main advantage layered approach simplicity construction debugging layer select use function operation layered operating system service low level layer approach simplify debugging system verification layer debug concern rest system definition use basic hardware assume correct implement function layer debug correct functioning assume second layer debug error find debugging particular layer error layer layer debug design implementation system simplify layer implement operation provide low level layer layer need know operation implement need know operation layer hide existence certain data structure operation hardware higher- layered system successfully computer network tcp ip web application relatively operate sys- tem use pure layered approach reason involve challenge appropriately define functionality layer addition overall performance system poor overhead require user program traverse multiple layer obtain operate system ser- vice layering common contemporary operating system generally system few layer functionality provide advantage modularized code avoid problem layer definition interaction see original unix system monolithic struc- ture unix expand kernel large difficult manage mid-1980 researcher carnegie mellon university develop operate system call mach modularize kernel micro- kernel approach method structure operate

call mach modularize kernel micro- kernel approach method structure operate system remove architecture typical microkernel nonessential component kernel implement user- level program reside separate address space result small kernel little consensus service remain kernel implement user space typically microkernel provide minimal process memory management addition communication facility figure 2.15 illustrate architecture typical main function microkernel provide communication client program service run user space communication provide message passing describe section 2.3.3.5 example client program wish access file interact file server client program service interact directly communicate indirectly exchange message microkernel benefit microkernel approach make extend operate system easy new service add user space conse- quently require modification kernel kernel modify change tend few microkernel small kernel result operate system easy port hardware design microkernel provide security reliability service run user kernel process service fail rest operate system remain untouched well know illustration microkernel operate system darwin kernel component macos ios operate system darwin fact consist kernel mach microkernel cover macos ios system detail section 2.8.5.1 example qnx real time operate system embed sys- tem qnx neutrino microkernel provide service message passing process scheduling handle low level network communication hardware interrupt service qnx provide standard pro- cesse run outside kernel user mode unfortunately performance microkernel suffer increase system function overhead user level service communicate message copy service reside separate address space addition operate system switch process exchange message overhead involve copy message switch process large impediment growth microkernel base operate system consider history windows nt release layered microkernel organi- zation version performance low compare windows 95 windows nt 4.0 partially correct performance problem move layer user space kernel space integrate closely time windows xp design windows architecture monolithic microkernel section 2.8.5.1 describe macos address performance issue mach microkernel good current methodology operating system design involve loadable kernel module lkms kernel set core

system design involve loadable kernel module lkms kernel set core component link additional service module boot time run time type design common modern implementation unix linux macos solaris windows idea design kernel provide core service service implement dynamically kernel run link service dynamically preferable add new feature directly kernel require recompile kernel time change example build cpu scheduling memory management algorithm directly kernel add support different file system way loadable module overall result resemble layer system kernel section define protect interface flexible layer system module module approach similar microkernel approach primary module core function knowledge load communicate module efficient module need invoke message passing order communicate linux use loadable kernel module primarily support device driver file system lkm insert kernel sys- tem start boot run time usb device plug run machine linux kernel nec- essary driver dynamically load lkm remove kernel run time linux lkm allow dynamic modular kernel maintain performance benefit monolithic system cover create lkms linux programming exercise end practice operate system adopt single strictly define struc- ture instead combine different structure result hybrid system address performance security usability issue example linux monolithic have operate system single address space provide efficient performance modular new functionality dynamically add kernel windows largely monolithic primarily performance reason retain behavior typical microkernel system include provide support separate subsystem know operating system personality run user mode process windows system provide support dynamically loadable kernel module provide case study linux windows 10 chapter 20 chapter 21 respectively remainder section explore structure hybrid system apple macos operat- e system prominent mobile operating system ios macos ios apple macos operate system design run primarily desktop laptop computer system ios mobile operate system design iphone smartphone ipad tablet computer architecturally macos ios common present highlight share differ general archi- tecture system show figure 2.16 highlight layer include

general archi- tecture system show figure 2.16 highlight layer include follow user experience layer layer define software interface allow user interact compute device macos use aqua user interface design mouse trackpad ios use springboard user interface design touch device application framework layer layer include cocoa cocoa touch framework provide api objective c swift programming language primary difference cocoa cocoa touch develop macos application ios provide support hardware feature unique mobile device touch screen core framework layer define framework support graphic medium include quicktime opengl kernel environment darwin architecture apple macos ios operating system kernel environment environment know darwin include mach microkernel bsd unix kernel elaborate show figure 2.16 application design advantage user experience feature bypass interact directly application framework core framework additionally application forego framework entirely communicate directly kernel environment example situation c program write user interface make posix system call significant distinction macos ios include follow- macos intend desktop laptop computer system compile run intel architecture ios design mobile device compile arm base architecture similarly ios ker- nel modify somewhat address specific feature need mobile system power management aggressive memory management additionally ios stringent security setting ios operating system generally restricted developer macos close developer example ios restrict access posix bsd api ios openly available developer macos focus darwin use hybrid structure darwin layered system consist primarily mach microkernel bsd unix kernel darwin structure show figure 2.17 operate system provide single system interface kernel standard c library unix linux system darwin provide system interface mach system call know structure darwin trap bsd system call provide posix functionality interface system call rich set library include standard c library library provide networking security progamming language support beneath system interface mach provide fundamental operating- system service include memory management cpu scheduling inter- process communication ipc facility message passing remote procedure call rpcs

process communication ipc facility message passing remote procedure call rpcs functionality provide mach available kernel abstraction include task mach process thread memory object port ipc example application create new process bsd posix fork system mach turn use task kernel abstraction represent process kernel addition mach bsd kernel environment provide o kit development device driver dynamically loadable module macos refer kernel extension kext section 2.8.3 describe overhead message passing different service run user space compromise performance microkernel address performance problem darwin combine mach bsd o kit kernel extension single address space mach pure microkernel sense subsystem run user space message passing mach occur copying necessary service access address space apple release darwin operate system open source result project add extra functionality darwin x- 11 windowing system support additional file system unlike darwin cocoa interface proprietary apple framework available develop macos application closed android operate system design open handset alliance lead primarily google develop android smartphone tablet computer ios design run apple mobile device close source android run variety mobile platform open- source partly explain rapid rise popularity structure android appear figure 2.18 android similar ios layered stack software provide rich set framework support graphic audio hardware feature feature turn provide platform develop mobile application run multitude android enable device software designer android device develop application java language generally use standard java api google design separate android api java development java application compile form execute android runtime art virtual machine design android optimize mobile device limited memory cpu processing capability java program compile java bytecode .class file translate executable .dex file java virtual machine perform time jit compilation improve application efficiency art perform ahead time aot compila- architecture google android tion .dex file compile native machine code instal device execute art aot compi- lation allow efficient application execution reduce power consumption feature

compi- lation allow efficient application execution reduce power consumption feature crucial mobile system android developer write java program use java native interface jni allow developer bypass virtual machine instead write java program access specific hardware feature program write jni generally portable hardware device set native library available android application include framework develop web browser webkit database support sqlite network support secure socket ssl android run unlimited number hardware device

google choose abstract physical hardware hard- ware abstraction layer hal abstract hardware camera gps chip sensor hal provide application consistent view independent specific hardware feature course allow devel- oper write program portable different hardware platform standard c library linux system gnu c library glibc google instead develop bionic standard c library android bionic small memory footprint glibc design slow cpu characterize mobile device addition bionic allow google bypass gpl licensing glibc android software stack linux kernel google modify linux kernel android variety area support special need mobile system power management change memory management allocation add new form ipc know binder cover section 3.8.2.1 window subsystem linux windows use hybrid architecture provide subsystem emu- late different operate system environment user mode subsystem communicate windows kernel provide actual service windows 10 add windows subsystem linux wsl allow native linux application specify elf binary run windows 10 typical operation user start windows application bash.exe present user bash shell run linux internally wsl create linux instance consist init process turn create bash shell run native linux application /bin bash process run windows pico process special process load native linux binary process address space provide environment linux application execute pico process communicate kernel service lxcore lxss translate linux system call possible native windows system call linux application make system windows equivalent lxss service provide equivalent functionality relationship linux windows system call lxss forward linux system directly equivalent windows kernel situation linux windows system call similar identical occur lxss provide functionality invoke similar windows system provide remainder functionality linux fork provide illustration windows createprocess system similar fork provide exactly functionality fork invoke wsl lxss service initial work fork call createprocess remainder work figure illustrate basic behavior wsl build boot operate system possible design code implement operating system specifically specific machine configuration commonly operating system design run

system specifically specific machine configuration commonly operating system design run class machine variety commonly computer system purchase operating system instal example purchase new laptop windows macos preinstalle suppose wish replace preinstalle oper- ating system add additional operate system suppose purchase computer operate system situation option place appropriate operate system computer configure use generate build operate system scratch follow step 1 write operate system source code obtain previously write 2 configure operate system system run 3 compile operate system 4 install operate system 5 boot computer new operate system configure system involve specify feature include vary operate system typically parameter describe system configure store configuration file type file create way extreme system administrator use modify copy operate system source code operate system completely compile know system build data declaration initialization constant compilation produce output object version operate system tailor system describe configuration slightly tailor level system description lead selec- tion precompile object module exist library module link form generate operate system process allow library contain device driver support o device need select link operate system system recompile system generation fast result system overly general support different hardware configuration extreme possible construct system completely modular selection occur execution time compile link time system generation involve simply set parameter describe system configuration build boot operating system major difference approach size generality generate system ease modify hardware configu- ration change embed system uncommon adopt approach create operate system specific static hardware config- uration modern operate system support desktop laptop computer mobile device adopt second approach operate system generate specific hardware config- uration use technique loadable kernel module provide modular support dynamic change system illusrate build linux system scratch typically necessary perform following step 1 download linux source code http://www.kernel.org 2 configure kernel menuconfig command step generate .config configuration file 3 compile main kernel command

step generate .config configuration file 3 compile main kernel command command compile kernel base configuration parameter identify .config file produce file vmlinuz kernel image 4 compile kernel module module command compile kernel module compilation depend con- figuration parameter specify .config file 5 use command module install install kernel mod- ule vmlinuz 6 install new kernel system enter install system reboot begin run new operate system alternatively possible modify exist system instal linux virtual machine allow host operate system windows macos run linux introduce virtualization section 1.7 cover topic fully chapter 18 option instal linux virtual machine alternative build virtual machine scratch option similar build linux system scratch operate system need compile approach use linux virtual machine appliance operate system build con- figure option simply require download appliance instal virtualization software virtualbox vmware example build operating system virtual machine provide text author following 1 download ubuntu iso image https://www.ubuntu.com/ 2 instruct virtual machine software virtualbox use iso bootable medium boot virtual machine 3 answer installation question instal boot operate system virtual machine operate system generate available use hardware hardware know kernel load kernel process start computer load kernel know boot system system boot process proceed 1 small piece code know bootstrap program boot loader locate kernel 2 kernel load memory start 3 kernel initialize hardware 4 root file system mount section briefly describe boot process detail computer system use multistage boot process computer power small boot loader locate nonvolatile firmware know bios run initial boot loader usually load second boot loader locate fixed disk location call boot block program store boot block sophisticated load entire operate system memory begin execution typically simple code fit single disk block know address disk length remainder bootstrap program recent computer system replace bios base boot process uefi unified extensible firmware interface uefi advantage bios include

process uefi unified extensible firmware interface uefi advantage bios include well support 64 bit system large disk great advantage uefi single complete boot manager fast multistage bios boot process boot bios uefi bootstrap program perform variety task addition load file contain kernel program memory run diagnostic determine state machine example inspect memory cpuand discover device diagnostic pass program continue booting step bootstrap initialize aspect system cpu register device controller content main memory soon later start operate system mount root file system point system say run grub open source bootstrap program linux unix system boot parameter system set grub configuration file load startup grub flexible allow change boot time include modify kernel parameter select different kernel boot example follow kernel parameter special linux file /proc cmdline boot time boot image kernel image load memory root specify unique identifier root file system save space decrease boot time linux kernel image compressed file extract load memory boot process boot loader typically create temporary ram file system know initramf file system contain necessary driver kernel module instal support real root file system main memory kernel start necessary driver instal kernel switch root file system temporary ram location appropriate root file system location finally linux create systemd process initial process system start service example web server and/or database ultimately system present user login prompt section 11.5.2 describe boot process worthwhile note booting mechanism independent boot loader specific version grub boot loader bios uefi firmware know specific bootloader boot process mobile system slightly different traditional pc example kernel linux base android use grub instead leave vendor provide boot loader common android boot loader lk little kernel android system use compressed kernel image linux initial ram file system linux discard initramf necessary driver load android maintain initramf root file system device kernel load root file system mount

root file system device kernel load root file system mount android start init process create number service display home screen finally boot loader operate system include windows linux macos ios android provide boot recovery mode single user mode diagnose hardware issue fix corrupt file system reinstall operate system addition hardware failure computer system suffer software error poor operating system performance consider follow section 2.10 operating system debugging mention debugging time time chapter close look broadly debugging activity find fix error system hardware software performance problem consider bug debugging include performance tuning seek improve performance remove processing bottleneck section explore debug process kernel error performance problem hardware debugging outside scope text process fail operate system write error information log fil alert system administrator user problem occur operate system core dump capture memory process store file later analysis memory refer core early day computing run program core dump probe debugger allow programmer explore code memory process time failure debug user level process code challenge operate system kernel debugging complex size complexity kernel control hardware lack user level debugging tool failure kernel call crash crash occur error information save log file memory state save crash dump operate system debugging process debugging frequently use dif- ferent tool technique different nature task consider kernel failure file system code risky kernel try save state file file system reboot common technique save kernel memory state section disk set aside purpose contain file system kernel detect unrecoverable error write entire content memory kernel own part system memory disk area system reboot process run gather datum area write crash dump file file system analysis obviously strategy unnecessary debug ordinary user level process performance monitoring tuning mention early performance tuning seek improve performance remove processing bottleneck identify bottleneck able monitor system performance operate system mean compute display measure system behavior

system performance operate system mean compute display measure system behavior tool characterize provide process system wide observation observation tool use approach counter tracing explore follow section operate system track system activity series counter number system call number operation perform network device disk following example linux tool use counter ps report information single process selection process report real time statistic current process vmstat report memory usage statistic netstat report statistic network interface iostat report o usage disk windows 10 task manager counter base tool linux system read statistic /proc file system /proc pseudo file system exist kernel memory primarily query process kernel statistic /proc file system organize directory hierarchy process unique integer value assign process appear subdirectory /proc example directory entry /proc/2155 contain process statistic process id 2155 /proc entry kernel statistic chapter chapter 3 provide programming project create access /proc file system windows system provide windows task manager tool include information current application process cpu memory usage networking statistic screen shot task manager windows 10 appear figure 2.19 counter based tool simply inquire current value certain statistic maintain kernel trace tool collect datum specific event step involve system invocation following example linux tool trace event strace trace system call invoke process gdb source level debugger perf collection linux performance tool tcpdump collect network packet debug twice hard write code place write code cleverly possible definition smart debug make operate system easy understand debug tune run active area research practice new generation kernel- enable performance analysis tool significant improvement goal achieve discuss bcc toolkit dynamic kernel tracing linux debug interaction user level kernel code nearly impossible toolset understand set code instru- ment interaction toolset truly useful able debug area system include area write debug- ging mind affect system reliability toolset minimal performance impact ideally impact use proportional impact use bcc toolkit

performance impact ideally impact use proportional impact use bcc toolkit meet requirement provide dynamic secure low impact debug bcc bpf compiler collection rich toolkit provide trace fea- ture linux system bcc end interface ebpf extended berkeley packet filter tool bpf technology develop early 1990 filter traffic computer network extended bpf ebpf add feature bpf ebpf program write subset c compile ebpf instruction dynamically insert run linux system ebpf instruction capture specific event certain system invoke monitor system per- formance time require perform disk o ensure ebpf instruction behave pass verifie insert run linux kernel verifier check sure instruction affect system performance security ebpf provide rich set feature tracing linux kernel traditionally difficult develop program c interface bcc develop easy write tool ebpf provide end interface python bcc tool write python embed c code interface ebpf instrumentation turn interface kernel bcc tool compile c program ebpf instruction insert kernel probe tracepoint technique allow trace event linux kernel specific write custom bcc tool scope text bcc package instal linux virtual machine provide provide number exist tool monitor area activity run linux kernel example bcc disksnoop tool trace disk o activity enter command generate following example output output tell timestamp o operation occur o read write operation byte involve o.

final column reflect duration express latency lat millisecond o. tool provide bcc specific application mysql database java python program probe place monitor activity specific process example ./opensnoop -p 1225 trace open system call perform process identifier bcc ebpf trace tool make bcc especially powerful tool live production system run critical application cause harm system particularly useful system administrator monitor system performance identify possible bottleneck security exploit figure 2.20 illustrate wide range tool currently provide bcc ebpf ability trace essentially area linux operat- e system bcc rapidly change technology new feature constantly operate system provide environment execution pro- grams provide service user program primary approach interact operate system 1 command interpreter 2 graphical user interface 3 touch- system call provide interface service available oper- ate system programmer use system application programming interface api access system service system call divide major category 1 process control 2 file management 3 device management 4 information maintenance 5 communication 6 protection standard c library provide system interface unix operate system include collection system program pro- vide utility user linker combine relocatable object module single binary executable file loader load executable file memory eligible run available cpu reason application operating system specific include different binary format program executable different instruction set different cpu system call vary operating system operate system design specific goal mind goal ultimately determine operate system policy operate system implement policy specific mechanism monolithic operating system structure functionality pro- vide single static binary file run single address space system difficult modify primary benefit layered operating system divide number discrete layer layer hardware interface high layer user interface layered software system success approach generally ideal design operate system performance problem microkernel approach design operate system use minimal kernel service run user level application communication take place message passing modular approach design operate system

communication take place message passing modular approach design operate system provide operating- system service module load remove run time contemporary operate system construct hybrid system combination monolithic kernel module boot loader load operate system memory perform initializa- tion begin system execution performance operate system monitor counter tracing counter collection system wide per- process statistic tracing follow execution program operate system purpose system call purpose command interpreter usually separate kernel system call execute command interpreter shell order start new process unix system purpose system program main advantage layered approach system design disadvantage layered approach list service provide operate system explain create convenience user case impossible user level program provide service explain answer system store operate system firmware store disk system design allow choice operating system boot bootstrap program need bryant o'hallaron 2015 provide overview computer system include role linker loader atlidakis et al 2016 discuss posix system call relate modern operate system levin 2013 cover internal macos ios levin 2015 describe detail android system windows 10 internal cover

russinovich et al 2017 bsd unix describe mckusick et al 2015 love 2010 mauerer 2008 thoroughly discuss linux kernel solaris fully describe mcdougall mauro 2007 linux source code available http://www.kernel.org ubuntu iso image available https://www.ubuntu.com/. comprehensive coverage linux kernel module find http://www.tldp.org/ldp/lkmpg/2.6/lkmpg.pdf ward 2015 http://www process grub performance tuning focus linux solaris system cover gregg 2014 detail bcc toolkit find atlidakis et al 2016 v. atlidakis j. andrus r. geambasu d. mitropoulos j. nieh posix abstraction modern operating systems old new missing 2016 page 19:1–19:17 bryant o'hallaron 2015 r. bryant d. o'hallaron computer systems programmer perspective edition 2015 b. gregg systems performance enterprise cloud pearson j. levin mac os x ios internals apple core wiley j. levin android internals confectioner cookbook volume r. love linux kernel development edition developer w. mauerer professional linux kernel architecture john wiley sons 2008 mcdougall mauro 2007 r. mcdougall j. mauro solaris internals second edition prentice hall 2007 mckusick et al 2015 m. k. mckusick g. v. neville neil r. n. m. wat- son design implementation freebsd unix operating system second edition pearson 2015 russinovich et al 2017 m. russinovich d. a. solomon a. ionescu win- dows internals 1 seventh edition microsoft press 2017 b. ward

linux work superuser know second edition starch press 2015 chapter 2 exercise service function provide operate system divide main category briefly describe category discuss differ describe general method pass parameter operate describe obtain statistical profile time program spend execute different section code discuss importance obtain statistical profile advantage disadvantage system- interface manipulate file device possible user develop new command interpreter system interface provide operate system describe android use ahead time aot time model interprocess communication strength weakness approach contrast compare application programming interface api application binary interface abi separation mechanism policy desirable difficult achieve layered approach component operate system dependent identify scenario unclear layer system component require tight coupling functionality main advantage microkernel approach system design user program system service interact micro- kernel architecture disadvantage microker- advantage loadable kernel module ios android similar different explain java program run android system use standard java api virtual machine experimental synthesis operate system assembler incor- porate kernel optimize system performance kernel assemble routine kernel space minimize path sys- tem kernel approach antithesis layered approach path kernel extend build operate system easy discuss pro con synthesis approach kernel design system performance section 2.3

describe program copy content file destination file program work prompt user source destination file write program posix windows api sure include necessary error checking include ensure source file exist correctly design test program system support run program utility trace sys- tem call linux system provide strace utility macos system use dtruss command dtruss command actually end dtrace require admin privilege run sudo tool follow assume executable file filecopy sudo dtruss ./filecopy windows system provide tool trace windows version program debugger introduction linux kernel module project learn create kernel module load linux kernel modify kernel module create entry /proc file system project complete linux virtual machine available text use text editor write c program use terminal application compile program enter command command line manage module kernel discover advantage develop kernel module relatively easy method interact kernel allow write program directly invoke kernel function important mind write kernel code directly interact kernel normally mean error code crash system virtual machine failure worst require reboot system i. kernel module overview project involve follow series step create insert module linux kernel list kernel module currently load enter command list current kernel module column size module function call module load int simple init(void printk(kern info load kernel modulen function call module remove void simple exit(void printk(kern info remove kernel modulen macro register module entry exit point module init(simple init module exit(simple exit module description("simple module kernel module simple.c program figure 2.21 name simple.c available source code text

illustrate basic kernel module print appropriate message load unload function simple init module entry point represent function invoke module load kernel simi- larly simple exit function module exit point function call module remove kernel module entry point function return integer value 0 represent success value represent failure module exit point function return void module entry point module exit point pass parameter follow macro register module entry exit point kernel module init(simple init module exit(simple exit notice figure module entry exit point function call printk function printk kernel equivalent printf output send kernel log buffer content read dmesg command difference printf printk printk allow specify priority flag value give < linux printk.h > include file instance priority kern info define informational message final line module license module description mod- ule author()—represent detail software license description module author purpose require infor- mation include standard practice develop kernel kernel module simple.c compile makefile accom- panye source code project compile module enter follow command line compilation produce file file simple.ko represent compile kernel module follow step illustrate insert module linux kernel ii load remove kernel module kernel module load insmod command run fol- sudo insmod simple.ko check module load enter lsmod command search module simple recall module entry point invoke module insert kernel check content message kernel log buffer enter command message loading module remove kernel module involve invoke rmmod command notice .ko suffix unnecessary sudo rmmod simple sure check dmesg command ensure module kernel log buffer fill quickly make sense clear buffer periodically accomplish follow sudo dmesg -c proceed step describe create kernel module load unload module sure check content kernel log buffer dmesg ensure follow step properly kernel module run kernel possible obtain value function available kernel regular user application example linux include file < linux hash.h > define hashing function use

include file < linux hash.h > define hashing function use kernel file define constant value golden ratio prime define unsigned long value print follow printk(kern info lun golden ratio prime example include file < linux gcd.h > define following unsigned long gcd(unsigned long unsigned b return great common divisor parameter b. able correctly load unload module complete following additional step 1 print value golden ratio prime simple init func- 2 print great common divisor 3,300 24 sim- ple exit function compiler error helpful perform kernel development important compile program run regularly sure load remove kernel module check kernel log buffer dmesg ensure change simple.c work properly section 1.4.3 describe role timer timer interrupt handler linux rate timer tick tick rate value hz define < asm param.h > value hz determine frequency timer interrupt value vary machine type architecture example value hz 100 timer interrupt occur 100 time second 10 millisecond additionally kernel keep track global variable jiffy maintain number timer interrupt occur system boot jiffy variable declare file < linux jiffies.h >

1 print value jiffy hz simple init function 2 print value jiffy simple exit function proceed set exercise consider use different value jiffy simple init simple exit determine number second elapse time kernel module load remove iii /proc file system /proc file system pseudo file system exist kernel mem- ory primarily query kernel process statistic include < linux proc fs.h > define buffer size 128 define proc hello ssize t proc read(struct file file char user usr buf size t count loff t po static struct file operation proc op = .owner = module .read = proc read function call module load int proc init(void create /proc hello entry proc create(proc 0666 null proc op function call module remove void proc exit(void remove /proc hello entry remove proc entry(proc null /proc file system kernel module 1 exercise involve design kernel module create additional entry /proc file system involve kernel statistic information relate specific process entire program include figure 2.22 figure begin describe create new entry /proc file sys- tem follow program example name hello.c available source code text create /proc entry name /proc hello user enter command infamous hello world message return function call time /proc hello read ssize t proc read(struct file file char user usr buf size t count loff t pos int rv = 0 char buffer[buffer size static int complete = 0 complete complete = 0 complete = 1

rv = sprintf(buffer hello worldn copy kernel space buffer user space usr buf copy user(usr buf buffer rv module init(proc init module exit(proc exit module description("hello module /proc file system kernel module 2 module entry point proc init create new /proc hello entry proc create function function pass proc op contain reference struct file operation struct initial- ize

.owner .read member value .read function proc read call /proc hello read examine proc read function string hello worldn write variable buffer buffer exist kernel mem- ory /proc hello access user space copy content buffer user space kernel function copy user function copy content kernel memory buffer variable usr buf exist user space time /proc hello file read proc read function call repeatedly return 0 logic ensure func- tion return 0 collect datum

case string hello worldn correspond /proc hello file finally notice /proc hello file remove module exit point proc exit function remove proc entry assignment involve design kernel module 1 design kernel module create /proc file name /proc jiffy report current value jiffy /proc jiffies file read command sure remove /proc jiffy module remove 2 design kernel module create proc file name /proc second report number elapsed second kernel module load involve value jiffy hz rate user enter command kernel module report number second elapse kernel module load sure remove /proc second module remove process program execution process need certain resource cpu time memory file o device accomplish task resource typically allocate process execute process unit work system system consist collection process operate system process execute system code user process execute user code process thread control system multiple hardware processing core thread run parallel important aspect operate system schedule thread available processing core choice design cpu scheduler available programmer c h p t e r early computer allow program execute time pro- gram complete control system access system resource contrast contemporary computer system allow multiple pro- gram load memory execute concurrently evolution require firm control compartmentalization pro- gram need result notion process program execution process unit work modern compute system complex operate system expect behalf user main concern execution user program need care system task well user space kernel system consist collection process execute user code execute operate system code potentially process execute concurrently cpu cpu multiplexe chapter read process represent operate system work identify separate component process illustrate represent schedule operate system describe process create terminate operate sys- tem include develop program appropriate system call perform operation describe contrast interprocess communication share memory message passing design program use pipe posix shared memory perform describe client server communication socket remote proce-

shared memory perform describe client server communication socket remote proce- design kernel module interact linux operate system question arise discuss operate system involve cpu activity early computer batch system execute job follow emergence time share system run user program task single user system user able run program time word processor web browser e mail package computer execute program time embedded device support multitasking operate system need support internal program activity memory management respect activity similar process personally prefer contemporary term process term job historical significance operating system theory terminology develop time major activity operate system job processing appropriate instance use job describe role operate system example mislead avoid use commonly accept term include word job job scheduling simply process supersede job informally mention early process program execution status current activity process represent value program counter content processor register memory layout process typically divide multiple section show figure 3.1 section include text section executable code datum section global variable layout process memory heap section memory dynamically allocate program run stack section temporary data storage invoke function function parameter return address local variable notice size text datum section fix size change program run time stack heap section shrink grow dynamically program execution time function call activation record contain function parameter local variable return address push stack control return function activation record pop stack similarly heap grow memory dynamically allocate shrink memory return system stack heap section grow operate system ensure overlap emphasize program process program passive entity file contain list instruction store disk call executable fil contrast process active entity program counter specify instruction execute set associate resource program process executable file load memory common technique load executable file double click icon represent executable file enter executable file command line prog.exe a.out process associate program consider separate execution sequence instance user run

process associate program consider separate execution sequence instance user run different copy mail program user invoke copy web browser program separate process text section equivalent data heap stack section vary common process spawn process run discuss matter section 3.4 note process execution environment code java programming environment provide good example cir- cumstance executable java program execute java virtual machine jvm jvm execute process interpret load java code take action native machine instruction behalf code example run compile java program program.class command java run jvm ordinary process turn execute java program program virtual machine concept simulation code instead write different instruction set write java language process execute change state state process define current activity process process following memory layout c program figure show illustrate layout c program memory highlight different section process relate actual c program figure similar general concept process memory show figure 3.1 difference global datum section divide different section initialize datum b uninitialized datum

separate section provide argc argv parameter pass main function int y = 15 int main(int argc char argv value = int malloc(sizeof(int)*5 for(i = 0 < 5 i++ values[i = gnu

size command determine size byte section assume executable file c program memory follow output generate enter command size memory datum field refer uninitialized datum bss refer initialize datum bss historical term refer block start symbol dec hex value sum section represent decimal new process create run instruction execute wait process wait event occur o completion reception signal ready process wait assign processor o event completion o event wait diagram process state terminate process finish execution name arbitrary vary operate system state represent find system certain operate sys- tem finely delineate process state important realize process run processor core instant process ready wait state diagram correspond state present figure 3.2 process control block process represent operate system process control block pcb)—also call task control block pcb show figure 3.3 contain piece information associate specific process process state state new ready run wait halt program counter counter indicate address instruction execute process list open file process control block pcb cpu register register vary number type depend computer architecture include accumulator index register stack pointer general purpose register plus condition code informa- tion program counter state information save interrupt occur allow process continue correctly afterward reschedule run cpu scheduling information information include process prior- ity pointer scheduling queue scheduling parameter chapter 5 describe process scheduling memory management information information include item value base limit register page table segment table depend memory system operate system chapter 9 accounting information information include cpu real time time limit account number job process number o status information information include list o device allocate process list open file brief pcb simply serve repository datum need start restart process accounting datum process model discuss far imply process program perform single thread execution example process run word processor program single thread instruction execute single thread control allow process perform task

thread instruction execute single thread control allow process perform task time user simultaneously type character run spell checker modern operate system extend process concept allow process multiple thread execution perform task time feature especially beneficial multicore system multiple thread run parallel multithreaded word processor example assign thread manage user input thread run spell checker system support thread pcb expand include information thread change system need support thread chapter 4 explore thread objective multiprogramming process run time maximize cpu utilization objective time sharing switch cpu core process frequently user interact program run meet objective process scheduler select available process possibly set available process program execution core cpu core run process time process representation linux resent c structure task struct find represent process include state process scheduling memory management information list open file pointer process parent list child sibling process parent process create child process create sibling child parent process field state process struct sched entity se scheduling information struct task struct parent process parent struct list head child process child struct file struct file list open file struct mm struct mm address space example state process represent field long state structure linux kernel active process represent doubly link list task struct kernel maintain pointer current process currently execute system show currently execute proccess illustration kernel manipulate field task struct specify process let assume system like change state process currently run value new state current pointer process currently execute state change follow current->state = new state system single cpu core process run time multicore system run multiple process time process core excess process ready queue wait queue wait core free reschedule number process currently memory know degree multiprogramming balance objective multiprogramming time sharing require take general behavior process account general process describe o bind cpu bind o bind computation cpu bind process

o bind cpu bind o bind computation cpu bind process contrast generate o request process enter system ready queue ready wait execute cpu core queue generally store link list ready queue header contain pointer pcb list pcb include pointer field point pcb ready system include queue process allocate cpu core execute eventually terminate interrupt wait occurrence particular event completion o request suppose process make o request device disk device run significantly slow processor process wait o available process wait certain event occur completion o place wait queue common representation process scheduling queueing diagram figure 3.5 type queue present ready queue set wait queue circle represent resource serve queue arrow indicate flow process system new process initially ready queue wait select execution dispatch process allocate cpu core execute event occur o wait queue wait queueing diagram representation process scheduling process issue o request place o wait process create new child process place wait queue await child termination process remove forcibly core result interrupt have time slice expire ready queue case process eventually switch waiting state ready state ready queue process continue cycle terminate time remove queue pcb resource deallocate process migrate ready queue wait queue through- lifetime role cpu scheduler select process ready queue allocate cpu core cpu scheduler select new process cpu frequently o bind process execute millisecond wait o request cpu bind process require cpu core long dura- tion scheduler unlikely grant core process extended period instead likely design forcibly remove cpu process schedule process run cpu scheduler execute 100 millisecond typically frequently operate system intermediate form scheduling know swapping key idea advantageous remove process memory active contention cpu reduce degree multiprogramming later process reintroduce memory execution continue leave scheme know swapping process swap memory disk current status save later swap disk memory status restore swapping typically necessary memory

later swap disk memory status restore swapping typically necessary memory overcommitte free swapping discuss chapter 9 mention section 1.2.1 interrupt cause operate system change cpu core current task run kernel routine operation happen frequently general purpose system interrupt occur system need save current context process run cpu core restore context processing essentially suspend process resume context represent pcb process include value cpu register process state figure 3.2 memory management information generically perform state save current state cpu core kernel user mode state restore resume operation switch cpu core process require perform state save current process state restore different process task know context switch illustrate figure 3.6 context switch occur kernel save context old process pcb load save context new process schedule run context- switch time pure overhead system useful work switch switching speed vary machine machine depend save state pcb0 save state pcb1 reload state pcb1 reload state pcb0 interrupt system interrupt system diagram show context switch process process multitaske mobile system constraint impose mobile device early version ios provide user application multitasking application run foreground user application suspend operating- system task multitaske write apple behave begin ios 4 apple provide limited form multitasking user application allow single foreground appli- cation run concurrently multiple background application mobile device foreground application application currently open appear display background application remain mem- ory occupy display screen ios 4 programming api provide support multitasking allow process run back- ground suspend limited available application type hardware mobile device begin offer large memory capacity multiple processing core great battery life subsequent version ios begin support rich functionality multi- tasking few restriction example large screen ipad tablet allow run foreground app time technique know origin android support multitasking place constraint type application run background application require processing background application use service separate application component run behalf background

background application use service separate application component run behalf background process consider stream audio application application move background service continue send audio datum audio device driver behalf background application fact service continue run background application suspend service user interface small memory footprint provide efficient technique multitaske mobile memory speed number register copy existence special instruction single instruction load store register typical speed microsecond context switch time highly dependent hardware support instance processor provide multiple set register context switch simply require change pointer current register set course active process register set system resort copy register datum memory complex operate system great work context switch chapter 9 advanced memory management technique require extra datum switch context instance address space current process preserve space task prepare use address space preserve work need preserve depend memory- management method operate system operation process process system execute concurrently cre- ate delete dynamically system provide mechanism process creation termination section explore mecha- nism involve create process illustrate process creation unix windows system course execution process create new process mention early create process call parent process new process call child process new process turn create process form tree process operate system include unix linux windows identify process accord unique process identifie pid typically integer number pid provide unique value process system index access attribute process kernel figure 3.7 illustrate typical process tree linux operate system show process pid use term process loosely situation linux prefer term task instead systemd process pid 1 serve root parent process user process user process create system boot system boot systemd process create process provide additional service web print server ssh server like figure 3.7 child systemd logind sshd logind process responsible manage client directly log system example client log bash shell assign pid 8416 bash command line interface user create process ps

pid 8416 bash command line interface user create process ps vim editor sshd process responsible manage client connect system ssh short secure shell pid = 2808 pid = 8415 pid = 1 pid = 8416 pid = 9298 pid = 9204 pid =

3028 pid = 3610

pid = 4005 tree process typical linux system operation process init systemd process traditional unix system identify process init root child process init know system v init assign pid 1 process create system boot process tree similar show figure 3.7 init root linux system initially adopt system v init approach recent distribution replace systemd describe section 3.3.1 systemd serve system initial process system v init flexible provide service unix linux system obtain listing process ps command example command list complete information process currently active system process tree similar show figure 3.7 construct recursively trace parent process way systemd process addition linux system provide pstree command display tree process system general process create child process child process need certain resource cpu time memory file o device accomplish task child process able obtain resource directly operate system constrain subset resource parent process parent partition resource child able share resource memory file child restrict child process subset parent resource prevent process overload system create child process addition supply physical logical resource parent process pass initialization datum input child process example consider process function display content file hw1.c screen terminal process create input parent process file hw1.c file open file write content output device alternatively operate system pass resource child process system new process open file hw1.c terminal device simply transfer datum process create new process possibility execution exist 1 parent continue execute concurrently child 2 parent wait child terminate address space possibility new process 1 child process duplicate parent process program datum parent 2 child process new program load illustrate difference let consider unix operate system unix see process identify process identifier unique integer new process create fork system new process consist copy address space original process mechanism allow parent process communicate easily child process process parent child continue execution instruction fork difference

child process process parent child continue execution instruction fork difference return code fork zero new child process nonzero process identifier child return parent fork system process typically use exec system replace process memory space new pro- gram exec system load binary file memory destroy memory image program contain exec system start pid t pid fork child process pid = fork pid < 0

error occur fprintf(stderr fork fail pid = = 0 child process parent process parent wait child complete create separate process unix fork system operation processes execution manner process able communicate separate way parent create child child run issue wait system ready queue termination child exec overlay process address space new program exec return control error occur c program show figure 3.8 illustrate unix system call pre- viously describe different process run copy program difference value variable pid child process zero parent integer value great zero fact actual pid child process child process inherit privilege scheduling attribute parent certain resource open file child process overlay address space unix command /bin ls directory listing execlp system execlp version exec system parent wait child process complete wait system child process complete implicitly explicitly invoke exit par- ent process resume wait complete exit system illustrate figure 3.9 course prevent child invoke exec instead continue execute copy parent process scenario parent child concurrent process run code instruction child copy parent process copy datum alternative example consider process creation windows process create windows api createprocess func- tion similar fork parent create new child process fork child process inherit address space parent createprocess require load specified program address space child process process creation furthermore fork pass parameter createprocess expect few c program show figure 3.10 illustrate createprocess function create child process load application mspaint.exe opt default value parameter pass cre- ateprocess reader interested pursue detail process creation management windows api encourage consult biblio- graphical note end chapter parameter pass createprocess function instance startupinfo process information structure startupinfo specify property new process window process creation fork system process information pi

allocate memory si.cb = sizeof(si create child process createprocess(null use command line c:windowssystem32mspaint.exe command null inherit process handle null inherit
thread handle false disable handle inheritance 0 creation flag null use parent environment block null use parent existing directory fprintf(stderr create process fail parent wait child complete close handle create separate process windows api size appearance handle standard

input output file process information structure contain handle identifier newly create process thread invoke zeromemory function allocate memory structure proceed parameter pass createprocess application command line parameter application null case command line parameter specify application load operation process instance load microsoft windows mspaint.exe appli- cation initial parameter use default parameter inherit process thread handle specify creation flag use parent exist environment block start directory provide pointer startupinfo process information structure create beginning program figure 3.8 parent process wait child complete invoke wait system equivalent windows waitforsingleobject pass handle child process pi.hprocess wait process complete child process exit control return waitforsingleobject function parent process process terminate finishes execute final statement ask operate system delete exit system point process return status value typically integer wait parent process wait system resource process include physical virtual memory open file o buffer deallocate reclaim operate system termination occur circumstance process cause termination process appropriate system example terminateprocess windows usually system invoke parent process terminate user misbehave application arbitrarily kill user process note parent need know identity child terminate process create new process identity newly create process pass parent parent terminate execution child variety reason child exceed usage resource allocate determine occur parent mechanism inspect state child task assign child long require parent exit operate system allow child continue parent terminate system allow child exist parent terminate system process terminate normally abnormally child terminate phenomenon refer cascade termination normally initiate operate system illustrate process execution termination consider linux

normally initiate operate system illustrate process execution termination consider linux unix system terminate process exit system provide exit status parameter exit status 1 fact normal termination exit call directly show indirectly c run time library add unix executable file include exit default parent process wait termination child process wait system wait system pass parameter allow parent obtain exit status child system return process identifier terminate child parent tell child terminate pid t pid pid = wait(&status process terminate resource deallocate operate system entry process table remain parent call wait process table contain process exit status process terminate parent call wait know zombie process process transition state terminate generally exist zombie briefly parent call wait process identifier zombie process entry process table release consider happen parent invoke wait instead terminate leave child process orphan traditional unix system address scenario assign init process new parent orphan process recall section 3.3.1 init serve root process hierarchy unix system init process periodically invoke wait allow exit status orphan process collect release orphan process identifier process table entry linux system replace init systemd process serve role linux allow process systemd inherit orphan process manage termination android process hierarchy resource constraint limited memory mobile operate system terminate existing process reclaim limited system resource terminate arbitrary process android identify importance hierarchy process system terminate process resource available new important process terminate process order increase importance important hierarchy process classification follow foreground process current process visible screen represent- e application user currently interact visible process process directly visible foreground perform activity foreground process refer process perform activity status display foreground process service process process similar background process perform activity apparent user streaming background process process perform activity apparent user process process hold active component associate application system resource reclaim android terminate process follow background process forth process assign

reclaim android terminate process follow background process forth process assign importance ranking android attempt assign process high ranking possible example process provide service visible assign important visible classification furthermore android development practice suggest follow guide- line process life cycle guideline follow state process save prior termination resume save state user navigate application process execute concurrently operate system inde- pendent process cooperate process process independent share datum process execute system process cooperate affect affect process execute system clearly process share datum process reason provide environment allow process information sharing application interested piece information instance copying pasting provide environment allow concurrent access information computation speedup want particular task run fast break subtask execute parallel notice speedup achieve computer multiple processing core modularity want construct system modular fashion divide system function separate process thread discuss chapter 2 cooperate process require interprocess communication ipc mechanism allow exchange datum send datum receive datum fundamental model interprocess communication share memory message passing share memory model region memory share cooperate process establish process exchange information read write datum share region message passing model multiprocess architecture chrome browser website contain active content javascript flash html5 provide rich dynamic web browsing experience unfortunately web application contain software bug result sluggish response time cause web browser crash big problem web browser display content web- site contemporary web browser provide tabbed browsing allow single instance web browser application open website time site separate tab switch dif- ferent site user need click appropriate tab arrangement illustrate aproblem approach web application tab crash entire process include tab display additional website crash google chrome web browser design address issue

multiprocess architecture chrome identify different type process browser renderer plug in browser process responsible manage user interface disk network o. new browser process create chrome start browser process create renderer process contain logic render web page contain logic handle html javascript image forth general rule new renderer process create website open new tab renderer process active plug process create type plug flash quicktime use plug process contain code plug additional code enable plug communicate associate renderer process browser process advantage multiprocess approach website run iso- lation website crash renderer process affect process remain unharmed furthermore renderer pro- cesse run sandbox mean access disk network o restrict minimize effect security exploit communication take place mean message exchange cooperate process communication model contrast ipc shared memory systems m0 m1 m2 communications model share memory b message passing model mention common operate system system implement message passing useful exchange small amount datum conflict need avoid message pass- ing easy implement distribute system share memory system provide distribute share memory consider text share memory fast message pass- ing message pass system typically implement system call require time consume task kernel intervention share memory system system call require establish shared- memory region share memory establish access treat routine memory access assistance kernel require section 3.5 section 3.6 explore shared memory message- pass system detail ipc shared memory systems interprocess communication share memory require communicate process establish region share memory typically share memory region reside address space process create share memory segment process wish communicate share memory segment attach address space recall normally oper- ating system try prevent process access process memory share memory require process agree remove restriction exchange information read write datum share area form datum location determine process operate system control pro- cesse responsible ensure write illustrate concept cooperate process let consider pro-

responsible ensure write illustrate concept cooperate process let consider pro- ducer consumer problem common paradigm cooperate pro- cesses producer process produce information consume con- sumer process example compiler produce assembly code consume assembler assembler turn produce object mod- ule consume loader producer consumer problem provide useful metaphor client server paradigm generally think server producer client consumer example web server produce provide web content html file image consume read client web browser request resource solution producer consumer problem use share memory allow producer consumer process run concurrently available buffer item fill producer empty consumer buffer reside region memory share producer consumer process producer produce item consumer consume item producer consumer synchronize consumer try consume item produce type buffer unbounded buffer place prac- tical limit size buffer consumer wait new item producer produce new item bounded buffer assume fix buffer size case consumer wait buffer producer wait buffer let look closely bound buffer illustrate interprocess communication share memory following variable reside region memory share producer consumer process define buffer size 10

typedef struct item buffer[buffer size int = 0 int = 0

share buffer implement circular array logical pointer variable point free position buffer point position buffer buffer = = buffer + 1 buffer size = = code producer process show figure 3.12 code consumer process show figure 3.13 producer process local variable produce new item produce store consumer process local variable consume item consume store scheme allow buffer size 1 item buffer time leave exercise provide solution buffer size item buffer time section 3.7.1 illustrate posix api share memory ipc message passing systems item produce true produce item produce + 1 buffer size = = buffer[in = produce = + 1 buffer size producer process share memory issue illustration address concern situation producer process consumer process attempt access share buffer concurrently chapter 6 chapter 7 discuss syn- chronization cooperate process implement effectively share memory environment ipc message passing systems section 3.5 show cooperate process communicate share memory environment scheme require process share region memory code access manipulate share memory write explicitly application programmer way achieve effect operate system provide mean cooperate process communicate message pass item consume true = = consume = buffer[out = + 1 buffer size consume item consume consumer process share memory message passing provide mechanism allow process communicate synchronize action share address space particularly useful distribute environment communicate process reside different computer connect network example internet chat program design chat participant communicate exchange message message pass facility provide operation message send process fixed variable size fixed sized message send system level implementation straight-forward restriction make task program diffi- cult conversely variable sized message require complex system level implementation programming task simple common kind tradeoff see operating system design process p q want communicate send message receive message communication link exist link implement variety way concern link physical implementation share memory hardware bus network cover chapter 19

physical implementation share memory hardware bus network cover chapter 19 logical implementation method logically implement link send()/receive operation direct indirect communication synchronous asynchronous communication automatic explicit buffering look issue relate feature process want communicate way refer use direct indirect communication direct communication process want communicate explicitly recipient sender communication scheme send receive primitive define send(p message)—send message process p. receive(q message)—receive message process q.

communication link scheme following property link establish automatically pair process want communicate process need know identity communicate ipc message passing systems link associate exactly process pair process exist exactly link scheme exhibit symmetry addressing sender pro- cess receiver process communicate variant scheme employ asymmetry addressing sender name recipient recipient require sender scheme send receive primitive define follows send(p message)—send message process p. receive(id message)—receive message process vari- able d set process communication disadvantage scheme symmetric asymmetric limited modularity result process definition change identifier process necessitate examine process definition reference old identifier find modify new identifier general hard coding technique iden- tifier explicitly state desirable technique involve indirection describe indirect communication message send receive mailbox port mailbox view abstractly object message place process message remove mailbox unique identification example posix message queue use integer value identify mailbox process com- municate process number different mailbox process communicate share mailbox send receive primitive define follows send(a message)—send message mailbox a. receive(a message)—receive message mailbox a. scheme communication link following property link establish pair process member pair share mailbox link associate process pair communicate process number different link exist link correspond mailbox suppose process p1 p2 p3 share mailbox a.

process p1 send message p2 p3 execute receive a. process receive message send p1 answer depend following method choose allow link associate process allow process time execute receive operation allow system select arbitrarily process receive mes- sage p2 p3 receive message system define algorithm select process receive message example round robin process turn receiv- e message system identify receiver sender mailbox own process operate system mailbox own process mailbox address space process distinguish owner receive message mailbox user send message mailbox mailbox unique owner confusion process receive message send mailbox process own mailbox terminate mailbox disappear process subsequently send message mailbox notify mailbox long exist contrast mailbox own operate system exis- tence independent attach particular process operate system provide mechanism allow process following create new mailbox send receive message mailbox delete mailbox process create new mailbox mailbox owner default initially owner process receive message mailbox ownership receiving privilege pass process appropriate system call course provision result multiple receiver mailbox communication process take place call send receive primitive different design option implement primitive message passing block nonblocking know synchronous asynchronous text encounter concept synchronous asynchronous behavior relation operate system algorithm block send send process block message receive receive process mailbox nonblocke send send process send message resume block receive receiver block message available nonblocke receive receiver retrieve valid message ipc message passing systems message produce true produce item produce producer process message passing different combination send receive possible send receive block rendezvous sender receiver solution producer consumer problem trivial use block send receive statement producer merely invoke block send wait message deliver receiver mailbox likewise consumer invoke receive block message available illustrate figure 3.14 3.15 communication direct indirect message exchange commu- nicate process reside temporary queue basically queue implement way zero capacity queue maximum length zero link message wait case sender

capacity queue maximum length zero link message wait case sender block recipient receive message bound capacity queue finite length n n message reside queue new message send message place queue message copy pointer message keep sender continue execution message consume true consume item consume consumer process message passing wait link capacity finite link sender block space available queue unbounded capacity queue length potentially infinite number message wait sender block zero capacity case refer message system buffering case refer system automatic buffering example ipc systems section explore different ipc system cover posix api share memory discuss message passing mach oper- ating system present windows ipc interestingly use share memory mechanism provide certain type message passing conclude pipe early ipc mechanism unix system posix shared memory ipc mechanism available posix system include share memory message passing explore posix api share posix share memory organize memory map file associate region share memory file process create share memory object shm open system follow fd = shm open(name o creat | o rdwr 0666 parameter specify share memory object process wish access shared memory refer object subsequent parameter specify share memory object cre- ate exist o creat object open read write o rdwr parameter establish file access permission share memory object successful shm open return integer file descriptor share memory object object establish ftruncate function configure size object byte set size object 4,096 byte finally mmap function establish memory map file contain share memory object return pointer memory map file access share memory object program show figure 3.16 figure 3.17 use producer consumer model implement share memory producer establish share memory object write share memory consumer read share memory example ipc systems size byte share memory object const int size = 4096 share memory object const char = os string write share memory const char

const char = os string write share memory const char message 0 = hello const char message 1 = world share memory file descriptor pointer share memory obect create share memory object fd = shm open(name o creat | o rdwr,0666 configure size share memory object memory map share memory object ptr = char mmap(0 size prot read | prot write map share fd 0

write share memory object ptr + = strlen(message 0 ptr + = strlen(message 1 producer process illustrate posix share memory api producer show figure 3.16 create share memory object name os write infamous string hello world share memory program memory map share memory object specify size allow write object flag map shared specify change share memory object visible process share object notice write share memory object call sprintf function write format string pointer ptr write increment pointer number byte write size byte share memory object const int size = 4096 share memory object const char = os share memory file descriptor pointer share memory obect open share memory object fd = shm open(name o rdonly 0666 memory map share memory object ptr = char mmap(0 size prot read | prot write map share fd 0 read share memory object remove share memory object

consumer process illustrate posix share memory api example ipc systems consumer process show figure 3.17 read output con- tents share memory consumer invoke shm unlink function remove share memory segment consumer access provide exercise posix share memory api programming exercise end chapter additionally provide detailed coverage memory mapping section 13.5 mach message passing example message passing consider mach operate system mach especially design distribute system show suitable desktop mobile system evidence inclusion macos ios operate system discuss chapter 2 mach kernel support creation destruction multiple task similar process multiple thread control few associate resource communication mach include inter- task communication carry message message send receive mailbox call port mach port finite size unidirectional way communication message send port response send separate reply port port multiple sender receiver mach use port represent resource task thread memory processor message passing provide object orient approach interact system resource service message passing occur port host separate host distribute system associate port collection port right identify capability necessary task interact port example task receive message port capability mach port right receive port task create port port owner owner task allow receive message port port owner manipulate capability port commonly establish reply port example assume task t1 own port p1 send message port p2 own task t2 t1 expect receive reply t2 grant t2 right mach port right send port p1 ownership port right task level mean thread belong task share port right thread belong task easily communicate exchange message thread port associate thread task create special port task self port notify port create kernel receive right task self port allow task send message kernel kernel send notification event occurrence task notify port course task receive right mach port allocate function create new port allocate space queue message identify right port port right represent port port

queue message identify right port port right represent port port access right port name simple integer value behave like unix file descriptor following example illustrate create port api mach port t port port right mach port allocate mach task self task refer mach port right receive right port port port right task access bootstrap port allow task register port create system wide bootstrap server port register bootstrap server task look port registry obtain right send message port queue associate port finite size initially message send port message copy queue message deliver reliably priority mach guarantee multiple message sender queue first- fifo order guarantee absolute ordering instance message sender queue order mach message contain following field fixed size message header contain metadata message include size message source destination port commonly send thread expect reply port source pass receive task use return address send reply variable sized body contain datum message simple complex simple message contain ordinary unstructured user datum interpret kernel complex message contain pointer memory location contain datum know line datum transfer port right task line datum pointer especially useful message pass large chunk datum simple message require copy package datum message line datum transmission require pointer refer memory location datum store function mach msg standard api send receive message value function parameter mach send msg mach rcv msg indicate send receive operation illustrate client task send simple message server task assume port client server associate client server task respectively code figure 3.18 show client task construct header send message server server task receive message send client mach msg function invoke user program perform message passing mach msg invoke function mach msg trap system mach kernel kernel mach msg trap call function mach msg overwrite trap handle actual passing message example ipc systems struct message mach msg header t header

example ipc systems struct message mach msg header t header mach port t client mach port t server client code struct message message construct header message.header.msgh size = sizeof(message message.header.msgh remote port = server message.header.msgh local port = client send message mach msg(&message.header message header mach send msg send message sizeof(message size message send 0 maximum size receive message unnecessary mach port null receive port unnecessary mach msg timeout time out mach port null notify port server code struct message message receive message mach msg(&message.header message header mach rcv msg send message 0 size message send sizeof(message maximum size receive message server receive port mach msg timeout time out mach port null notify port example program illustrate message passing mach send receive operation flexible instance message send port queue queue message copy queue sending task continue port queue sender option specify parameter mach msg 1 wait indefinitely room queue 2 wait n millisecond 3 wait return immediately 4 temporarily cache message message give operate system queue message send message queue notification message send sender message queue pende time give send thread final option mean server task finish request server task need send time reply task request service continue service request reply port client major problem message system generally poor perfor- mance cause copying message sender port receiver port mach message system attempt avoid copy operation virtual memory management technique chapter 10 essentially mach map address space contain sender message receiver address space message actually copy sender receiver access memory message management technique provide large performance boost work intrasystem message windows operate system example modern design employ modularity increase functionality decrease time need imple- ment new feature windows provide support multiple operate envi- ronment subsystem application program communicate sub- system message passing mechanism application program consider client subsystem server message pass

passing mechanism application program consider client subsystem server message pass facility windows call advanced local pro- cedure alpc facility communication process machine similar standard remote procedure rpc mechanism widely optimize specific windows remote procedure call cover detail section 3.8.2 like mach win- dows use port object establish maintain connection process windows use type port connection port communi- server process publish connection port object visible pro- cesse client want service subsystem open handle server connection port object send connection request port server create channel return handle client chan- nel consist pair private communication port client server message server client message additionally communication channel support callback mechanism allow client server accept request normally expect reply example ipc systems > 256 byte advanced local procedure call windows alpc channel create message passing technique 1 small message 256 byte port message queue intermediate storage message copy process 2 large message pass section object region share memory associate channel 3 datum large fit section object api available allow server process read write directly address space client client decide set channel need send large message client determine want send large message ask section object create similarly server decide reply large create section object section object small message send contain pointer size information section object method complicated method list avoid data copying structure advanced local procedure call windows show figure 3.19 important note alpc facility windows windows api visible application programmer application windows api invoke standard remote procedure call rpc invoke process system rpc handle indirectly alpc procedure additionally kernel service use alpc communicate client process pipe act conduit allow process communicate pipe ipc mechanism early unix system typically pro- vide simple way process communicate limitation implement pipe issue consider 1 pipe allow bidirectional communication communication 2 way communication allow half duplex data travel way time duplex

way communication allow half duplex data travel way time duplex datum travel direction time 3 relationship parent child exist communi- 4 pipe communicate network communicate process reside machine follow section explore common type pipe unix windows system ordinary pipe name pipe ordinary pipe allow process communicate standard producer consumer fashion producer write end pipe write end consumer read end read end result ordinary pipe unidirectional allow way communication way communication require pipe pipe send datum different direction illustrate construct ordinary pipe unix windows system program example process write message greeting pipe process read message pipe unix system ordinary pipe construct function function create pipe access int fd file descrip- tor fd[0 read end pipe fd[1 write end unix treat pipe special type file pipe access ordinary read write system call ordinary pipe access outside process create typically parent process create pipe use communicate child process create fork recall section 3.3.1 child process inherit open file parent pipe special type file child inherit pipe parent process figure 3.20 illustrate file descriptor ordinary pipe example ipc systems define buffer size 25 define read end 0 define write end 1

char write msg[buffer size = greeting char read msg[buffer size pid t pid program continue figure 3.22 ordinary pipe unix relationship file descriptor fd array parent child process illustrate write parent write end pipe fd[1]—can read child read end fd[0]—of unix program show figure 3.21 parent process create pipe send fork create child process occur fork depend datum flow pipe instance parent write pipe child read important notice parent process child process initially close unused end pipe program show figure 3.21 require action important step ensure process read pipe detect end file read return 0 writer close end pipe ordinary pipe windows system term anonymous pipe behave similarly unix counterpart unidirectional employ parent child relationship communicate process addition read write pipe accomplish ordi- nary readfile writefile function windows api create pipe createpipe function pass parameter parameter provide separate handle 1 reading 2 writing pipe 3 instance startupinfo structure specify child process inherit handle pipe furthermore 4 size pipe byte specify figure 3.23 illustrate parent process create anonymous pipe communicate child unlike unix system child pro- cess automatically inherit pipe create parent windows require programmer specify attribute child process inherit create pipe pipe(fd = = -1 fork child process pid = fork pid < 0 error occur fprintf(stderr fork fail pid > 0 parent process close unused end pipe write pipe write(fd[write end write msg strlen(write msg)+1 close

write end pipe child process close unused end pipe read pipe read(fd[read end read msg buffer size printf("read s",read msg close read end pipe figure 3.21 continue accomplish initialize security attribute structure allow handle inherit redirect child process handle standard input standard output read write handle pipe child read pipe parent redirect child standard input read handle pipe furthermore pipe half duplex necessary prohibit child inherit write end examples ipc systems define buffer size 25 handle readhandle writehandle process information pi char message[buffer size = greeting program continue figure 3.24 windows anonymous pipe parent process pipe program create child process similar program figure 3.10 fifth parameter set true indicate child process inherit designate handle parent write pipe parent close unused read end pipe child process read pipe show figure 3.25 read pipe program obtain read handle pipe invoke getstdhandle note ordinary pipe require parent child relationship communicate process unix windows system mean pipe communication process ordinary pipe provide simple mechanism allow pair process communicate ordinary pipe exist process communicate unix windows system process finish communicate terminate ordinary pipe cease exist name pipe provide powerful communication tool com- munication bidirectional parent child relationship require name pipe establish process use communi- cation fact typical scenario name pipe writer addi- tionally name pipe continue exist communicate process finish unix windows system support name pipe detail implementation differ greatly explore name pipe system name pipe refer fifos unix system create appear typical file file system fifo create mkfifo system manipulate ordinary open read write close system call continue exist explicitly delete set security attribute allow pipe inherit security attribute sa = sizeof(security attributes),null true allocate memory create pipe createpipe(&readhandle writehandle sa 0 fprintf(stderr create pipe fail establish start info structure

child process si.hstdoutput = getstdhandle(std output handle redirect standard input read end pipe si.hstdinput = readhandle si.dwflags = startf usestdhandle allow child inherit write end pipe sethandleinformation(writehandle handle flag inherit 0 create child process createprocess(null child.exe null null true inherit handle 0 null null si pi close unused end pipe parent write pipe writefile(writehandle message buffer size,&written null fprintf(stderr error writing pipe close write end pipe wait child exit figure 3.23 continue communication client server systems define buffer size 25 char buffer[buffer size read handle pipe readhandle = getstdhandle(std input handle child read pipe readfile(readhandle buffer buffer size read null printf("child read s",buffer fprintf(stderr error reading pipe windows anonymous pipe child process file system fifo allow bidirectional communication half duplex transmission permit datum travel direction

fifo typically additionally communicate process reside machine intermachine communication require sock- et section 3.8.1 name pipe windows system provide rich communication mech- anism unix counterpart duplex communication allow communicate process reside different machine additionally byte orient datum transmit unix fifo windows system allow byte- message orient datum name pipe create createnamedpipe function client connect name pipe connectnamedpipe communi- cation name pipe accomplish readfile communication client server system section 3.4 describe process communicate share memory message passing technique communica- tion client server system section 1.10.3 section explore strategy communication client server system socket pipe practice pipe unix command line environment situ- ation output command serve input example unix ls command produce directory listing especially long directory listing output scroll screen command manage output display screen output time user use certain key forward backward file set pipe ls command run individual process allow output ls deliver input enable user display large directory list screen time pipe construct command line | character complete command ls | scenario ls command serve producer output consume command windows system provide command dos shell func- tionality similar unix counterpart unix system provide command tongue cheek style common unix command fact provide functionality dos shell use | character establish pipe difference directory listing dos use dir command ls dir |

remote procedure call rpcs shall coverage rpc useful client server computing android use remote procedure form ipc process run system asocket define endpoint communication apair process com- municate network employ pair socket process socket identify ip address concatenate port number general socket use client server architecture server wait incoming client request listen specify port request receive server accept connection client socket complete connection server implement specific service ssh ftp http listen know port ssh server listen port 22 ftp server listen port 21 web http server listen port 80 port 1024 consider know implement standard service client process initiate request connection assign port host computer port arbitrary number great 1024 example client host x ip address 146.86.5.20 wish establish connection web server listen port 80 communication client server systems communication socket address 161.25.19.8 host x assign port 1625 connection consist pair socket 146.86.5.20:1625 host x 161.25.19.8:80 web server situation illustrate figure 3.26 packet travel host deliver appropriate process base destination port number connection unique process host x wish establish connection web server assign port number great 1024 equal 1625 ensure connection consist unique pair socket program example text use c illustrate socket java provide easy interface socket rich library networking utility interested socket programming c c++ consult bibliographical note end chapter java provide different type socket connection orient tcp socket implement socket class connectionless udp socket use datagramsocket class finally multicastsocket class sub- class datagramsocket class multicast socket allow datum send example describe date server use connection orient tcp socket operation allow client request current date time server server listen port 6013 port arbitrary unused number great 1024 connection receive server return date time client date server show figure 3.27

server create serversocket specify listen port 6013 server begin listen port accept method server block accept method wait client request connection connection request receive accept return socket server use communicate client detail server communicate socket follow server establish printwriter object use communi- cate client printwriter object allow server write socket routine print println method output public class dateserver public static void main(string arg serversocket sock = new serversocket(6013 listen connection true socket client = sock.accept printwriter pout = new write date socket close socket resume listen connection catch ioexception ioe server process send date client call method println write date socket server close socket client resume listen request client communicate server create socket connect port server listen implement client java program show figure 3.28 client create socket request connection server ip address 127.0.0.1 port 6013 connection client read socket normal stream o statement receive date server client close socket exit ip address 127.0.0.1 special ip address know loopback computer refer ip address 127.0.0.1 refer mechanism allow client server host communicate tcp ip protocol ip address 127.0.0.1 replace ip address host run date server addition ip address actual host www.westminstercollege.edu communication client server systems public class dateclient public static void main(string arg connection server socket socket sock = new socket("127.0.0.1",6013 inputstream = sock.getinputstream bufferedreader bin = new read date socket line = bin.readline = null close socket connection*/ catch ioexception ioe

communication socket common efficient con- sidere low level form communication distribute process reason socket allow unstructured stream byte exchange communicate thread responsibility client server application impose structure datum sub- section look high level method communication remote procedure remote procedure calls common form remote service rpc paradigm design way abstract procedure mechanism use system network connection similar respect ipc mechanism describe section 3.4 usually build system deal environment process execute separate system use message base communication scheme provide remote service contrast ipc message message exchange rpc communi- cation structure long packet datum message address rpc daemon listen port remote sys- tem contain identifier specify function execute parameter pass function function execute request output send requester separate message port context simply number include start message packet system normally network address port address differentiate network service support remote process need service address message proper port instance system wish allow system able list current user daemon support rpc attach port port 3027 remote system obtain need information list current user send rpc message port 3027 server datum receive reply message semantic rpc allow client invoke procedure remote host invoke procedure locally rpc system hide detail allow communication place provide stub client typically separate stub exist separate remote procedure client invoke remote procedure rpc system call appropriate stub pass parameter provide remote procedure stub locate port server marshal parameter stub transmit message server message passing similar stub server receive message invoke procedure server necessary return value pass client technique windows system stub code compile specification write microsoft interface definitio language midl define interface client server program parameter marshaling address issue concern difference datum representation client server machine consider representa- tion 32 bit integer system know big endian store significant byte system know little endian store significant byte order well se choice arbitrary computer architecture resolve

significant byte order well se choice arbitrary computer architecture resolve difference like rpc system define machine independent representation datum representation know external datum representation xdr client parameter marshaling involve convert machine dependent datum xdr send server server xdr datum unmarshale convert machine dependent representation important issue involve semantic local procedure call fail extreme circumstance rpc fail duplicate execute result common network error way address problem operate system ensure message act exactly local procedure call exactly functionality difficult

consider semantic implement attach- e timestamp message server history timestamp message process history large communication client server systems ensure repeat message detect income message timestamp history ignore client send message time assure execute exactly need remove risk server receive request accomplish server implement protocol describe acknowledge client rpc receive execute ack message common networking client resend rpc periodically receive ack important issue concern communication server client standard procedure call form binding take place link load execution time chapter 9 procedure replace memory address procedure rpc scheme require similar binding client server port client know port number server system information share memory approach common bind information prede- termine form fix port address compile time rpc fix port number associate program compile server change port number request service second binding dynamically rendezvous mechanism typically operate system provide rendezvous call matchmaker daemon fixed rpc port aclient send message contain rpc rendezvous daemon request port address rpc need execute port number return rpc call send port process terminate server crash method require extra overhead initial request flexible approach figure 3.29 show sample interaction rpc scheme useful implement distribute file system chap- ter 19 system implement set rpc daemon client message address distribute file system port server file operation place message contain disk operation perform disk operation read write rename delete status correspond usual file relate system call return message contain datum result exe- cut dfs daemon behalf client instance message contain request transfer file client limit simple block request case request need file transfer rpc typically associate client server computing dis- tributed system form ipc process run system android operate system rich set ipc mechanism contain binder framework include rpc allow process request service process android define application component basic building block provide utility android application app combine multiple application component provide functionality app

android application app combine multiple application component provide functionality app applica- user call kernel send rpc reply client port p port p receive find port number rpc x port port p rpc x port p user user execution remote procedure rpc tion component service user interface instead run background execute long run operation perform work remote process example service include play music back- ground retrieve datum network connection behalf process prevent process block datum download client app invoke bindservice method service service bound available provide client server communication message passing rpc bind service extend android class service imple- ment method onbind invoke client call bindser- vice case message passing onbind method return mes- senger service send message client service messenger service way service send reply client client provide messenger service contain replyto field message object send service service send message client provide rpc onbind method return interface repre- sente method remote object client use interact service interface write regular java syntax use android interface definition language aidl create stub file serve client interface remote service briefly outline process require provide generic remote service name remotemethod aidl binder service inter- face remote service appear follow remoteservice.aidl boolean remotemethod(int x double y file write remoteservice.aidl android development kit use generate .java interface .aidl file stub serve rpc interface service server implement interface generate .aidl file implementation interface call client invoke remotemethod client call bindservice onbind method invoke server return stub remoteservice object client client invoke remote method follow internally android binder framework handle parameter marshaling transfer marshal parameter process invoke nec- essary implementation service send return value client process aprocess program execution status current activity process represent program counter register layout process memory represent different section 1 text 2 datum 3 heap 4 stack process execute change state general state process 1

4 stack process execute change state general state process 1 ready 2 run 3 wait 4 terminate process control block pcb kernel datum structure represent process operate system role process scheduler select available process run operate system perform context switch switch run process run fork createprocess system call create pro- cesse unix windows system respectively share memory communication process process share region memory posix provide api share memory process communicate exchange message message passing mach

operate system use message passing primary form interprocess communication windows provide form message passing pipe provide conduit process communicate form pipe ordinary name ordinary pipe design communication process parent child relationship name pipe general allow process communi- unix system provide ordinary pipe pipe system ordinary pipe read end write end parent process example send datum pipe write end child process read read end name pipe unix term fifos windows system provide form pipe anonymous name pipe anonymous pipe similar unix ordinary pipe unidirectional employ parent child relationship communicate process name pipe offer rich form interprocess communication unix counterpart fifos common form client server communication socket remote procedure call rpcs socket allow process different machine communicate network rpc abstract concept function procedure call way function invoke process reside separate computer android operate system use rpc form interprocess com- munication binder framework program show figure 3.30 explain output line a. include initial parent process process create program show figure 3.31 original version apple mobile ios operate system provide means concurrent processing discuss major complication concurrent processing add operate system computer system provide multiple register set describe happen context switch occur new context int value = 5 pid t pid pid = fork pid = = 0

child process value + = 15 pid > 0 parent process printf("parent value = d",value line output line load register set happen new context memory register set register set process create new process fork operation following state share parent process child share memory segment consider exactly once”semantic respect rpc mechanism algorithm implement semantic execute correctly ack message send client lose net- work problem describe sequence message discuss exactly preserve assume distribute system susceptible server failure mechanism require guarantee exactly semantic execution rpc fork child process fork child process fork process create process creation management ipc unix windows system respectively discuss robbins robbins 2003 russinovich et al 2017 love 2010 cover support process linux kernel hart

2005 cover windows system programming detail coverage multiprocess model google chrome find message passing multicore system discuss holland seltzer 2011 levin 2013 describe message passing mach system particu- larly respect macos ios harold 2005 provide coverage socket programming java detail android rpcs find https://developer.android.com/guide/compo nents aidl.html hart 2005 robbins robbins 2003 cover pipe windows unix system respectively guideline android development find https://developer.and e. r. harold java network programming edition o'reilly associates 2005 j. m. hart windows system programming edition addison- holland seltzer 2011 d. holland m. seltzer multicore os look- e forward 1991 er 2011 proceeding 13th usenix conference hot topic operating system 2011 page 33–33 j. levin mac os x ios internals apple core wiley r. love linux kernel development edition developer robbins robbins 2003 k. robbins s. robbins unix systems pro- gramming communication concurrency thread second edition prentice russinovich et al 2017 m. russinovich d. a. solomon a. ionescu win- dows internals 1

seventh edition microsoft press 2017 chapter 3 exercise describe action take kernel context switch pro- construct process tree similar figure 3.7 obtain process infor- mation unix linux system use command ps -ael use command man ps information ps com- mand task manager windows system provide parent process id process monitor tool available tech- net.microsoft.com provide process tree tool explain role init systemd process unix linux system regard process termination include initial parent process process create program show figure 3.32 explain circumstance line code mark printf("line j figure 3.33 reach program figure 3.34 identify value pid line b c d. assume actual pid parent child 2600 2603 respectively example situation ordinary pipe suitable name pipe example situation name pipe suitable ordinary pipe consider rpc mechanism describe undesirable consequence arise enforce exactly semantic describe possible use mechanism guarantee program show figure 3.35 explain output line x y. = 0 < 4 i++ process create benefit disadvantage follow consider system level programmer level synchronous asynchronous communication automatic explicit buffering send copy send reference fixed sized variable sized message pid t pid fork child process pid = fork pid < 0 error occur fprintf(stderr fork fail pid = = 0 child process parent process parent wait child complete line j reach pid t pid pid1 fork child process pid = fork pid < 0 error occur fprintf(stderr fork fail pid = = 0 child process pid1 = getpid printf("child pid = d",pid printf("child pid1 = d",pid1 b parent process pid1 = getpid printf("parent pid = d",pid c printf("parent pid1 = d",pid1 d pid value define size 5 int nums[size = 0,1,2,3,4 pid t pid pid = fork pid = = 0 = 0 < size i++ nums[i = -i printf("child d nums[i line x pid > 0 = 0

< size i++ printf("parent d nums[i line y output line x line y unix linux system write c program fork child process ultimately zombie process zombie process remain system 10 second process state obtain command process state show s column process state z zombie process identifier pid child process list pid column parent list ppid column easy way determine child process zombie run program write background run command ps -l determine child zombie process want zombie process exist system need remove create easy way terminate parent process kill command example pid parent 4884

enter kill -9 4884 write c program call time.c determine time necessary run command command line program run ./time < command > report elapsed time run specify command involve fork exec function gettimeofday function deter- elapsed time require use different ipc general strategy fork child process execute specify command child execute command record timestamp current time term starting time parent process wait child process terminate child terminate parent record current timestamp end time difference start end time represent elapsed time execute command example output report time run command ls elapsed time 0.25422 parent child separate process need arrange start time share write version program represent different method version child process write starting time region share memory call exec child process terminate parent read starting time share memory refer section 3.7.1 detail posix shared memory section separate program producer consumer solution problem require single program region share memory establish child process fork allow parent child process access region second version use pipe child write start time pipe parent read follow termination child process use gettimeofday function record current timestamp function pass pointer struct timeval object contain member tv sec t usec repre- send number elapse second microsecond january 1 1970 know unix epoch following code sample illustrate function struct timeval current current.tv sec represent second current.tv usec represent microsecond ipc child parent process content shared memory pointer assign struct timeval repre- sente start time pipe pointer struct timeval write read pipe operate system pid manager responsible manage process identifier process create assign unique pid pid manager pid return pid manager process complete execution manager later reassign pid process identifier discuss fully section 3.3.1 important recognize process identifier unique active process pid use following constant identify range possible pid define min pid 300 define max pid 5000 use data structure choice

pid 300 define max pid 5000 use data structure choice represent avail- ability process identifier strategy adopt linux use bitmap value 0 position indicate process d value available value 1 indicate process d currently use implement following api obtain release pid int allocate map(void)—creates initialize datum struc- ture represent pid return 1 unsuccessful 1 successful int allocate pid(void)—allocate return pid return 1

unable allocate pid pid use void release pid(int pid)—release pid programming problem modify later chapter 4 chapter 6 collatz conjecture concern happen posi- tive integer n apply following algorithm n 3 × n + 1 n odd conjecture state algorithm continually apply positive integer eventually reach 1 example n = 35 35 106 53 160 80 40 20 10 5 16 8 4 2 1 write c program fork system generate sequence child process start number provide command line example 8 pass parameter command line child process output 8 4 2 1 parent child process copy datum necessary child output sequence parent invoke wait wait child process complete exit program perform necessary error checking ensure positive integer pass command line exercise 3.21 child process output sequence num- ber generate algorithm specify collatz conjecture parent child copy datum approach design program establish share memory object parent child process technique allow child write content sequence share memory object parent output sequence child com- plete memory share change child make reflect parent process program structure posix share memory describe section 3.7.1 parent process progress establish share memory object shm open ftruncate create child process wait terminate output content share memory remove share memory object area concern cooperate process involve synchro- nization issue exercise parent child process coordinate parent output sequence child finish execution process synchronize wait system parent process invoke wait suspend child process exit section 3.8.1 describe certain port number know provide standard service port 17 know quote the- day service client connect port 17 server server respond quote day modify date server show figure 3.27 deliver quote day current date quote printable ascii character contain few 512 character multiple line allow know port reserved unavailable server listen port 6017 date client show figure 3.28 read quote return ahaiku line poem line contain syllable second line contain seven syllable line contain syllable write haiku server listen port 5575 client connect port server respond haiku date client show figure 3.28 read quote return echo server echo receive client exam- ple client send server string hello server respond hello write echo server java networking api describe section 3.8.1 server wait client connection accept method client connection receive server loop perform following step read datum socket buffer write content buffer client server break loop determine client close connection date server figure 3.27

use java.io bufferedreader class bufferedreader extend java.io reader class read character stream echo server guarantee read character client receive binary datum class java.io inputstream deal datum byte level character level echo server use object extend java.io inputstream read method java.io inputstream class return 1 client close end socket connection design program ordinary pipe process send string message second process second process reverse case character message send process example process send message hi second process return hi require pipe send original message second process send modify message second process write program unix design file copy program name filecopy.c ordinary pipe program pass parameter file copy destination file program create ordinary pipe write content file copy pipe child process read file pipe write destination file example invoke program follow ./filecopy input.txt copy.txt file input.txt write pipe child process read content file write destination file copy.txt write program unix windows pipe project 1 unix shell project consist design c program serve shell interface accept user command execute command separate process implementation support input output redirection pipe form ipc pair command complete project involve unix fork exec wait dup2 pipe system call complete linux unix macos shell interface give user prompt command enter example illustrate prompt osh >

user command cat prog.c command display file prog.c terminal unix cat command technique implement shell interface parent process read user enter command line case cat prog.c create separate child process perform command specify parent process wait child exit contin- uing similar functionality new process creation illustrate figure 3.9 unix shell typically allow child process run background concurrently accomplish add ampersand end command rewrite command osh > cat prog.c parent child process run concurrently separate child process create fork system user command execute system call exec family describe section 3.3.1 c program provide general operation command line shell supply figure 3.36 main function present prompt osh- > outline step take input user read main function continually loop long run equal 1 user enter exit prompt program set run 0 define max line 80 maximum length command char args[max line/2 + 1 command line argument int run = 1 flag determine exit program run read user input step 1 fork child process fork 2 child process invoke execvp 3 parent invoke wait command include outline simple shell project organize part 1 create child process execute command child 2 provide history feature 3 add support input output redirection 4 allow parent child process communicate pipe ii execute command child process task modify main function figure 3.36 child process fork execute command specify user require parse user enter separate token store token array character string arg figure 3.36 example user enter command ps -ael osh > prompt value store arg array args[0 = ps args[1 = -ael args[2 = null arg array pass execvp function follow- execvp(char command char params command represent command perform param store parameter command project execvp function invoke execvp(args[0 args sure check

user include determine parent process wait child exit iii create history feature task modify shell interface program provide history feature allow user execute recent command enter example user enter command ls -l execute command enter prompt command execute fashion echo user screen command place history buffer command program manage basic error handling recent command history enter result message command iv redirect input output shell modify support > < redirection operator > redirect output command file < redirect input command file example user enter osh > l > out.txt output ls command redirect file out.txt simi- larly input redirect example user enter osh > sort < in.txt file in.txt serve input sort command manage redirection input output involve dup2 function duplicate exist file descriptor file descriptor example fd file descriptor file out.txt dup2(fd stdout fileno duplicate fd standard output terminal mean write standard output fact send out.txt file assume command contain input output redirection contain word concern command sequence sort < in.txt > out.txt v. communication pipe

final modification shell allow output command serve input pipe example follow command osh > ls -l | output command ls -l serve input com- mand ls command run separate process communicate unix pipe function describe section 3.7.4 easy way create separate process parent process create child process execute ls -l child create child process execute establish pipe child process create implement pipe functionality require dup2 function describe previous section finally command chain multiple pipe assume command contain pipe character combine redirection operator project 2 linux kernel module task information project write linux kernel module use /proc file system display task information base process identifier value pid begin project sure complete linux kernel module programming project chapter 2 involve create entry /proc file system project involve write process identifier file /proc pid pid write /proc file subsequent read /proc pid report 1 command task run 2 value task pid 3 current state task example kernel module access load system follow echo 1395 > /proc pid command = bash pid = 1395 state = 1 echo command write character 1395 /proc pid file kernel module read value store integer equivalent rep- resent process identifier cat command read /proc pid kernel module retrieve field task struct associ- ate task pid value 1395 ssize t proc write(struct file file char user usr buf size t count loff t po int rv = 0 char k mem allocate kernel memory k mem = kmalloc(count gfp kernel copy user space usr buf kernel memory copy user(k mem usr buf count printk(kern info sn k mem return kernel memory proc write function i. write /proc file system kernel module project chapter 2 learn read /proc file system cover write /proc set field .write struct file operation .write = proc

write cause proc write function figure 3.37 call write operation /proc pid kmalloc function kernel equivalent user level mal- loc function allocate memory kernel memory allocate gfp kernel flag indicate routine kernel memory allocation copy user function copy content usr buf con- tain write /proc pid recently allocate kernel memory kernel module obtain integer equivalent value kernel function kstrtol signature int kstrtol(const char str unsigned int base long re store character equivalent str express base

finally note return memory previously allocate kmalloc kernel kfree careful memory man- agement include release memory prevent memory leak crucial develop kernel level code ii read /proc file system process identifier store read /proc pid return command process identifier state illustrate section 3.1 pcb linux represent structure task struct find < linux sched.h > include file give process identifier function pid task return associate task struct signature function appear follow struct task struct pid task(struct pid pid enum pid type type kernel function find vpid(int pid obtain struct pid pidtype pid pid type valid pid system pid task return task struct display value command pid state probably read task struct structure < linux sched.h > obtain name field pid task pass valid pid return null sure perform appropriate error checking check condition situation occur kernel module function associate read /proc pid source code download c program pid.c pro- vide basic building block begin project project 3 linux kernel module list task project write kernel module list current task linux system iterate task linearly depth iterate task linearly linux kernel process macro easily allow iteration current task system struct task struct task process(task iteration task point task field task struct display program loop process macro design kernel module iterate task system process macro particular output task command state process d task probably read task struct structure < linux sched.h > obtain name field write code module entry point content appear kernel log buffer view dmesg command verify code work correctly compare content kernel log buffer output following command list task value similar task dynamic possible task appear listing ii iterate task depth search tree second portion project involve iterate task system depth search dfs tree example dfs iteration process figure 3.7 1 8415 8416 9298 9204 2808 3028 3610 4005

linux maintain process tree series list examine task struct < linux sched.h > struct list head object object pointer list task child sibling linux maintain reference initial task system init task type task struct information macro operation list iterate child init task follow struct task struct task struct list head list list each(list init task->children task = list entry(list struct task struct sibling task point child list list macro pass parameter type struct pointer head list traverse pointer head node list traverse iteration list parameter set list structure child use value obtain structure list list entry macro begin init task task design kernel module iterate task system dfs tree project output state pid task perform iteration kernel entry module output appear kernel log buffer output task system task appear ps -ael command thread appear child ordinary process check output dfs tree use command command list task include thread system verify perform appropriate dfs iteration examine relationship task output ps command project 4 kernel data structure section 1.9 cover data structure common oper- ating system linux kernel provide structure explore circular doubly link list available kernel developer discuss available linux source code instance include file < linux list.h>—and recommend examine file proceed following step initially define struct contain element insert link list following c struct define color mixture red blue green struct color struct list head list notice member struct list head list list head structure define include file < linux types.h > intention embed link list node comprise list list head structure simple merely hold member prev point previous entry list embed link list structure linux make possible manage data structure series i. insert element linked list declare list head object use reference head list list head macro static list head(color list macro define initialize variable color list type struct

head(color list macro define initialize variable color list type struct list head create initialize instance struct color follow struct color violet violet = kmalloc(sizeof(*violet gfp kernel violet->red = 138 violet->blue = 43 violet->green = 226

init list head(&violet->list kmalloc function kernel equivalent user level malloc function allocate memory kernel memory allocate gfp kernel flag indicate routine kernel memory allocation macro init list head initialize list member struct color add instance end link list list add tail macro list add tail(&violet->list color list ii traverse linked list traverse list involve list entry macro accept parameter pointer structure iterate pointer head list iterate variable contain list head structure following code illustrate macro struct color ptr list entry(ptr color list list iteration ptr point struct color iii remove element linked list remove element list involve list del macro pass pointer struct list head list del(struct list head element remove element list maintain structure remainder list simple approach remove element link list remove element traverse list macro list entry safe behave like list entry pass additional argument maintain value pointer item delete necessary preserve structure list following code example illustrate macro struct color ptr list entry safe(ptr next,&color list list iteration ptr point struct color notice delete element return memory previously allocate kmalloc kernel kfree module entry point create link list contain struct color element traverse link list output content kernel log buffer invoke dmesg command ensure list properly con- structe kernel module load module exit point delete element link list return free memory kernel invoke dmesg command check list remove kernel module unload ii parameter passing portion project involve pass parameter kernel module module use parameter initial value generate collatz sequence describe exercise 3.21 pass parameter kernel module parameter pass kernel module load exam- ple kernel module collatz pass initial value 15 kernel parameter start follow sudo insmod collatz.ko start=15

kernel module declare start parameter following static int start = 25 module param(start int 0 module param macro establish variable parameter kernel module module param provide argument 1 parameter 2 type 3 file permission file system access parameter concern permission use default value 0 note parameter insmod command match associate kernel parameter finally provide value module parameter load insmod default value case 25 design kernel module name collatz pass

initial value module parameter module generate store sequence kernel link list module load sequence store module traverse list output content kernel log buffer use dmesg command ensure sequence properly generate module load module exit point delete content list return free memory kernel use dmesg check list remove kernel module unload c h p t e r process model introduce chapter 3 assume process execute program single thread control virtually modern operat- e system provide feature enable process contain multiple thread control identify opportunity parallelism use thread increasingly important modern multicore system provide multiple cpu chapter introduce concept challenge associ- ate multithreaded computer system include discussion api pthreads windows java thread library additionally explore new feature abstract concept create thread allow developer focus identify opportunity parallelism let language feature api framework manage detail thread creation management look number issue relate multithreaded pro- gramming effect design operate system finally explore windows linux operate system support thread kernel identify basic component thread contrast thread describe major benefit significant challenge design multi- illustrate different approach implicit threading include thread pool fork join grand central dispatch describe windows linux operate system represent design multithreaded application pthread java windows threads concurrency thread basic unit cpu utilization comprise thread id program counter pc register set stack share thread belong process code section data section operate system resource open file signal traditional process single thread control process multiple thread control perform task time figure 4.1

illustrate difference traditional single threaded process multithreaded process software application run modern computer mobile device multithreaded application typically implement separate pro- cess thread control highlight example application create photo thumbnail collection image use separate thread generate thumbnail separate aweb browser thread display image text thread retrieve datum network word processor thread display graphic thread respond keystroke user thread perform spelling grammar checking background single threaded multithreaded process application design leverage processing capability mul- ticore system application perform cpu intensive task parallel multiple compute core certain situation single application require perform sev- eral similar task example web server accept client request web page image sound forth busy web server per- haps thousand client concurrently access web server run traditional single threaded process able service client time client wait long time request solution server run single process accept request server receive request create separate process service request fact process creation method common use thread popular process creation time consume resource intensive new process perform task exist process incur overhead generally efficient use process contain multiple thread web server process multithreade server create separate thread listen client request request create process server create new thread service request resume listen additional request illustrate figure 4.2 operate system kernel typically multithreaded example system boot time linux system kernel thread create thread perform specific task manage device memory management interrupt handling command ps -ef display kernel thread run linux system examine output command kernel thread kthreadd pid = 2 serve parent kernel thread application advantage multiple thread include basic sorting tree graph algorithm addition programmer solve contemporary cpu intensive problem data mining graphic artificial intelligence leverage power modern multicore system design solution run parallel 2 create new thread service 3 resume listen multithreaded server architecture thread concurrency benefit multithreaded programming break 1 responsiveness multithreade interactive application

concurrency benefit multithreaded programming break 1 responsiveness multithreade interactive application allow program continue run block perform- e lengthy operation increase responsiveness user quality especially useful design user interface instance consider happen user click button result performance time consume operation single threaded appli- cation unresponsive user operation complete contrast time consume operation perform separate asynchronous thread application remain responsive 2 resource sharing process share resource technique share memory message passing technique explicitly arrange programmer thread share memory resource process belong default benefit share code datum allow application different thread activity address space 3 economy allocate memory resource process creation costly thread share resource process belong economical create context switch thread empirically gauge difference overhead difficult general thread creation consume time memory process creation additionally context switching typically fast thread process 4 scalability benefit multithreading great mul- tiprocessor architecture thread run parallel different processing core single threaded process run processor regardless available explore issue follow section early history computer design response need computing performance single cpu system evolve multi cpu system later similar trend system design place multiple compute core single process chip core appear separate cpu operate system section 1.3.2 refer system multicore multithreaded programming provide mechanism efficient use multiple compute core improved concurrency consider application thread system single compute core concurrency merely mean execution thread interleave time figure 4.3 process core capable execute thread time system multiple core concurrency concurrent execution single core system mean thread run parallel system assign separate thread core figure 4.4 notice distinction concurrency parallelism discus- sion aconcurrent system support task allow task progress contrast parallel system perform task simultaneously possible concurrency parallelism advent multiprocessor multicore architecture com- puter system single processor cpu scheduler design provide illusion parallelism rapidly switch process allow process progress process run concurrently parallel trend multicore system continue place pressure system designer

concurrently parallel trend multicore system continue place pressure system designer application programmer well use multiple compute core designer operate system write scheduling algo- rithm use multiple processing core allow parallel execution show figure 4.4 application programmer challenge modify existing program design new program multithreaded general area present challenge program multicore 1 identify task involve examine application find area divide separate concurrent task ideally task independent run parallel individual 2 balance identify task run parallel programmer ensure task perform equal work equal value instance certain task contribute value overall process task separate execution core run task worth cost parallel execution multicore system threads concurrency amdahl law formula identify potential performance gain add additional compute core application serial nonparallel parallel component s portion application perform serially system n processing core formula appear follow s + 1s example assume application 75 percent parallel 25 percent serial run application system processing core speedup 1.6 time add additional core total speedup 2.28 time graph illustrate amdahl law different scenario number processing core s = 0.05 s = 0.10 s = 0.50 interesting fact amdahl law n approach infinity speedup converge 1s.

example 50 percent application perform serially maximum speedup 2.0 time regardless number processing core add fundamental principle amdahl law serial portion application dispropor- tionate effect performance gain add additional computing 3 datum splitting application divide separate task datum access manipulate task divide run 4 data dependency datum access task examine dependency task task depend datum programmer ensure execution task synchronize accommodate datum dependency examine strategy chapter 6 datum task parallelism 5 testing debugging program run parallel multi- ple core different execution path possible test debug- ging concurrent program inherently difficult test debug single threaded application challenge software developer argue advent multicore system require entirely new approach design software system future similarly computer science educator believe software development teach increase emphasis parallel type parallelism general type parallelism datum parallelism task par- allelism datum parallelism focus distribute subset datum multiple compute core perform operation core consider example sum content array size n. single core system thread simply sum element 0 n 1 dual core system thread run core 0 sum element 0 n2 1 thread b run core 1 sum element n2 n 1 thread run parallel

separate compute core task parallelism involve distribute datum task thread multiple compute core thread perform unique operation dif- ferent thread operate datum operate different datum consider example contrast situation example task parallelism involve thread perform unique statistical operation array element thread operate parallel separate compute core perform fundamentally datum parallelism involve distribution datum multiple core task parallelism involve distribution task multiple core show figure 4.5 datum task paral- threads concurrency user kernel thread lelism mutually exclusive application fact use hybrid strategy discussion far treat thread generic sense support thread provide user level user thread kernel kernel thread user thread support kernel manage kernel support kernel thread support manage directly operate system virtually contemporary operate system include windows linux macos support kernel ultimately relationship exist user thread kernel thread illustrate figure 4.6 section look common way establish relationship model to- model model model figure 4.7 map user level thread kernel thread thread management thread library user space efficient discuss thread library section 4.4 entire process block thread make block system thread access kernel time multiple thread unable run parallel multicore system green thread thread library available solaris system adopt early version java to- model system continue use model inability advantage multiple processing core standard computer system model figure 4.8 map user thread kernel thread provide concurrency model allow thread run thread make block system allow mul- tiple thread run parallel multiprocessor drawback model create user thread require create corresponding kernel thread large number kernel thread burden performance system linux family windows operate system imple- ment model model figure 4.9 multiplexe user level thread small equal number kernel thread number kernel thread specific particular application particular machine application allocate kernel thread system processing core system core thread concurrency let consider effect design concurrency many- model allow developer create user thread wish result

concurrency many- model allow developer create user thread wish result parallelism kernel schedule kernel thread time model allow great concurrency developer careful create thread application fact system limit number thread create model suffer shortcoming developer create user thread necessary correspond kernel thread run parallel multiprocessor thread perform block system kernel schedule thread execution variation model multiplexe user- level thread small equal number kernel thread allow user level thread bind kernel thread variation refer level model figure 4.10 model appear flexible model discuss practice difficult implement addition increase number processing core appear system limit number kernel thread important result operate system use model shall section 4.5 contemporary concurrency library developer identify task map thread model thread library provide programmer api create man- age thread primary way implement thread library approach provide library entirely user space kernel support code datum structure library exist

user space

mean invoke function library result local function user space system second approach implement kernel level library support directly operate system case code datum structure library exist kernel space invoke function api library typically result system kernel main thread library use today posix pthreads windows java pthread thread extension posix standard provide user level kernel level library windows thread library kernel level library available windows system java thread api allow thread create manage directly java program instance jvm run host operate system java thread api generally implement thread library available host system mean windows system java thread typ- ically implement windows api unix linux macos system typically use pthread posix windows threading datum declare globally declare outside function share thread belong process java equivalent notion global datum access share datum explicitly arrange thread remainder section describe basic thread creation thread library illustrative example design multi- threaded program perform summation non negative integer separate thread know summation function example n 5 function represent summation integer 1 5 15 program run upper bound summation enter command line user enter 8 summation integer value 1 8 output proceed example thread creation introduce general strategy create multiple thread asynchronous threading synchronous threading asynchronous threading parent create child thread parent resume execution parent child execute concurrently independently thread independent typically little datum sharing asynchronous threading strategy multithreaded server illustrate figure 4.2 commonly design responsive synchronous threading occur parent thread create child wait child terminate resume thread create parent perform work concurrently parent continue work complete thread finish work terminate join parent child join parent resume execution typically synchronous threading involve significant datum sharing thread example parent thread combine result calculate child follow example use synchronous threading pthread refer posix standard ieee 1003.1c define api thread creation synchronization specification thread behavior implementation operate system designer implement specification

synchronization specification thread behavior implementation operate system designer implement specification thread concurrency int sum data share thread(s

void runner(void param thread function int main(int argc char argv pthread t tid thread identifier pthread attr t attr set thread attribute set default attribute thread pthread attr init(&attr create thread pthread create(&tid attr runner argv[1 wait thread exit printf("sum = dn",sum thread execute function void runner(void param int upper = atoi(param sum = 0 = 1 < = upper i++ sum + = multithreaded c program pthreads api way wish numerous system implement pthreads specifica- tion unix type system include linux macos win- dows support pthreads natively party implementation windows available c program show figure 4.11 demonstrate basic pthreads api construct multithreaded program calculate summation non negative integer separate thread pthreads program separate thread begin execution specify function figure 4.11 run- ner function program begin single thread control begin define num threads 10 array thread join pthread t workers[num thread int = 0 < num thread i++

pthread join(workers[i null pthread code join thread main initialization main create second thread begin control runner function thread share global datum sum let look closely program pthreads program include pthread.h header file statement pthread t tid declare identifier thread create thread set attribute include stack size scheduling information pthread attr t attr declaration represent attribute thread set attribute function pthread attr init(&attr explicitly set attribute use default attribute provide chapter 5 discuss scheduling attribute provide pthreads api separate thread create pthread create function addi- tion pass thread identifier attribute thread pass function new thread begin execution case runner function pass integer parameter provide command line argv[1 point program thread initial parent thread main summation child thread perform summation oper- ation runner function program follow thread create join strategy create summation thread parent thread wait terminate call pthread join function summa- tion thread terminate call function pthread exit summation thread return parent thread output value share datum sum example program create single thread grow dominance multicore system write program contain thread increasingly common simple method wait thread pthread join function enclose operation simple loop example join thread pthread code show figure 4.12 technique create thread windows thread library similar pthreads technique way illustrate windows thread api c program show figure 4.13 notice include windows.h header file windows api thread concurrency dword sum data share thread(s thread execute function dword winapi summation(lpvoid param dword upper = dword*)param dword = 1 < = upper i++ sum + = int main(int argc char argv param = atoi(argv[1 create thread threadhandle = createthread null default security attribute 0 default stack size summation thread function param parameter thread function 0 default creation flag threadid return thread identifier wait thread finish close thread handle printf("sum = dn",sum multithreaded c program windows api pthread version show figure 4.11

datum share

separate thread case sum declare globally dword datum type unsigned 32 bit integer define summation function perform separate thread function pass pointer void windows define lpvoid thread perform function set global datum sum value summation 0 parameter pass summation thread create windows api createthread func- tion pthreads set attribute thread pass function attribute include security information size stack flag set indicate thread start suspended state program use default value attribute default value initially set thread suspend state instead eligible run cpu scheduler summation thread create parent wait complete output value sum value set summation thread recall pthread program figure 4.11 parent thread wait summation thread pthread join statement perform equivalent windows api waitforsingleobject function cause create thread block summation thread exit situation require wait multiple thread complete waitformultipleobjects function function pass 1 number object wait 2 pointer array object 3 flag indicate object signal 4 timeout duration infinite example thandles array thread handle object size n parent thread wait child thread complete statement waitformultipleobjects(n thandles true infinite thread fundamental model program execution java program java language api provide rich set feature creation management thread java program comprise single thread control simple java program consist main method run single thread jvm java thread available system provide jvm include windows linux macos java thread api available android application technique explicitly create thread java program approach create new class derive thread class override run method alternative commonly technique define class implement runnable interface interface define single abstract method signature public void run code run method class implement runnable execute separate thread example show class task implement runnable public void run system.out.println("i thread thread concurrency lambda expression java begin version 1.8 language java introduce lambda expres- sion allow clean syntax create thread define separate class implement runnable lambda expression instead runnable task = >

system.out.println("i thread thread worker = new thread(task lambda expression similar function know closure prominent feature functional programming language available nonfunctional language include python c++ c shall later example chapter lamdba expression provide simple syntax develop parallel application thread creation java involve create thread object pass instance class implement runnable follow invoke start method thread object appear following example thread worker = new thread(new task invoke start method new thread object thing 1 allocate memory initialize new thread jvm 2 call run method make thread eligible run jvm note run method directly start method call run method behalf recall parent thread pthreads windows library use pthread join waitforsingleobject respectively wait summation thread finish proceed join method java provide similar functionality notice join throw interrupt- edexception choose ignore catch interruptedexception ie parent wait thread finish join method enclose loop similar show pthreads figure 4.12 java executor framework java support thread creation approach describe far origin begin version 1.5 api java introduce new concurrency feature provide developer great control thread creation communication tool available java.util.concurrent package explicitly create thread object thread creation instead organize executor interface public interface executor void execute(runnable command class implement interface define execute method pass runnable object java developer mean executor create separate thread object invoke start method executor follow executor service = new executor executor framework base producer consumer model task implement runnable interface produce thread exe- cute task consume advantage approach divide thread creation execution provide mechanism communication concurrent task datum sharing thread belong process occur easily windows pthread share datum simply declare globally pure object orient language java notion global datum pass parameter class implement runnable java thread return result address need java.util.concurrent pack- age additionally define callable interface behave similarly runnable result return result return callable task know future object result retrieve method define

return callable task know future object result retrieve method define future interface program

show figure 4.14 illustrate summation program java feature summation class implement callable interface specify method v call()—it code method execute separate thread execute code create newsinglethreadex- ecutor object provide static method executor class type executorservice pass callable task submit method primary difference execute submit meth- od return result return result future submit callable task thread wait result call method future object return easy notice model thread creation appear complicated simply create thread join termination incur modest degree complication confer benefit see callable future allow thread return result threads

concurrency class summation implement callable < integer > private int upper public summation(int upper this.upper = upper thread execute method public integer int sum = 0 int = 1 < = upper i++ sum + = return new integer(sum public class driver public static void main(string arg int upper = integer.parseint(args[0 executorservice pool = executors.newsinglethreadexecutor future < integer >

result = pool.submit(new summation(upper system.out.println("sum = + result.get catch interruptedexception | executionexception ie illustration java executor framework api additionally approach separate creation thread result produce wait thread terminate retrieve result parent instead wait result available finally shall section 4.5.1 framework combine feature create robust tool manage large number thread continue growth multicore processing application contain- ing hundred thousand thread loom horizon design application trivial undertaking programmer jvm host operate system jvm typically implement host operate system figure 18.10 setup allow jvm hide implementation detail underlie operate system provide consistent abstract environment allow java program operate platform support jvm specification jvm indicate java thread map underlie operating system instead leave decision particular implementation jvm example windows operate system use model java thread jvm run windows map kernel thread addition relationship java thread library thread library host operate system example implementation jvm windows family operate system use windows api create java thread linux macos system use pthread address challenge outline section 4.2 additional difficul- tie difficulty relate program correctness cover chapter 6 chapter 8 way address difficulty well support design con- current parallel application transfer creation management threading application developer compiler run time library strategy term implicit threading increasingly popular trend section explore alternative approach design application advantage multicore processor implicit threading shall strategy generally require application developer identify task thread run parallel task usually writ- function run time library map separate thread typically model section 4.3.3 advantage approach developer need identify parallel task library determine specific detail thread creation management section 4.1 describe multithreaded web server situation server receive request create separate thread service request create separate thread certainly superior create separate process multithreaded server nonetheless potential problem issue concern time require create thread fact thread discard complete work second issue troublesome allow concurrent request service new thread place bound number

troublesome allow concurrent request service new thread place bound number thread concurrently active system unlimited thread exhaust system resource cpu time memory solution problem use thread pool thread concurrency android thread pool section 3.8.2.1 cover rpc android operate system recall section android use android interface defi- nition language aidl tool specify remote interface client interact server aidl provide thread pool aremote service thread pool handle multiple concurrent request service request separate thread pool general idea thread pool create number thread start place pool sit wait work server receive request create thread instead submit request thread pool resume wait additional request available thread pool awaken request service immediately pool contain available thread task queue free thread complete service return pool await work thread pool work task submit pool execute asynchronously thread pool offer benefit 1 service request exist thread fast wait create thread 2 athread pool limit number thread exist point particularly important system support large number concurrent thread 3 separate task perform mechanic create task allow use different strategy run task example task schedule execute time delay execute number thread pool set heuristically base factor number cpu system physical memory expect number concurrent client request sophisticated thread- pool architecture dynamically adjust number thread pool accord usage pattern architecture provide benefit have small pool consume memory load system low discuss architecture apple grand central dispatch later section windows api provide function relate thread pool thread pool api similar create thread thread create function describe section 4.4.2 function run separate thread define function appear follow dword winapi poolfunction(pvoid param function run separate thread pointer poolfunction pass function thread pool api thread pool execute function member thread pool api queueuserworkitem function pass lpthread start routine function pointer function run separate thread pvoid param parameter pass function ulong flag flag indicate thread pool create

parameter pass function ulong flag flag indicate thread pool create man- age execution thread example invoke function follow queueuserworkitem(&poolfunction null 0 cause thread thread pool invoke poolfunction behalf programmer instance pass parameter poolfunc- tion specify 0 flag provide thread pool special instruction thread creation member windows thread pool api include utility invoke function periodic interval asynchronous o request java thread pools java.util.concurrent package include api variety thread pool architecture focus follow model 1 single thread executor newsinglethreadexecutor()—creates pool size 1 2

fixed thread executor newfixedthreadpool(int size)—create thread pool specify number thread unbounded thread pool reuse thread instance fact see use java thread pool section 4.4.3 create newsinglethreadexecutor program example show figure 4.14 section note java executor frame- work construct robust threading tool describe create thread pool athread pool create factory method executors static executorservice newsinglethreadexecutor static executorservice newfixedthreadpool(int size static executorservice newcachedthreadpool factory method create return object instance imple- ment executorservice interface executorservice extend execu- thread concurrency public class threadpoolexample public static void main(string arg int numtask = integer.parseint(args[0].trim create thread pool executorservice pool = executors.newcachedthreadpool run task thread pool int = 0

< numtask i++ shut pool thread complete create thread pool java tor interface allow invoke execute method object addition executorservice provide method manage termination thread pool example show figure 4.15 create cache thread pool submit task execute thread pool execute method shutdown method invoke thread pool reject additional task shut exist task complete execution strategy thread creation cover section 4.4 know fork join model recall method main parent thread create fork child thread wait child terminate join point retrieve combine result synchronous model characterize explicit thread creation excellent candidate implicit threading situation thread construct directly fork stage parallel task designate model illustrate figure 4.16 library manage number thread create responsible assign task thread way fork join model synchronous version thread pool library determine actual number thread create example heuristic describe section 4.5.1 fork join java java introduce fork join library version 1.7 api design recursive divide conquer algorithm quicksort mergesort implement divide conquer algorithm library separate task fork divide step assign small subset original problem algorithm design separate task execute concurrently point size problem assign task small solve directly require create additional task general recursive algorithm java fork join model show problem small solve problem directly subtask1 = fork(new task(subset problem subtask2 = fork(new task(subset problem result1 = join(subtask1 result2 = join(subtask2 return combine result figure 4.17 depict model graphically illustrate java fork join strategy design divide and- conquer algorithm sum element array integer version 1.7 api java introduce new thread pool forkjoinpool assign task inherit abstract base class forkjointask assume sumtask

class follow create forkjoin- pool object submit initial task invoke method forkjoinpool pool = new forkjoinpool array contain integer sum int array = new int[size sumtask task = new sumtask(0 size 1 array int sum = pool.invoke(task completion initial invoke return summation array class sumtask show figure 4.18

implement divide and- conquer algorithm sum content array fork join new task create fork method compute method speci- fie computation perform task method compute invoke directly calculate sum subset assign threads concurrency fork join java join block task complete join return result calculate compute notice sumtask figure 4.18 extend recursivetask java fork- join strategy organize abstract base class forkjointask recursivetask recursiveaction class extend class fun- damental difference class recursivetask return result return value specify compute recursiveaction return result relationship class illustrate uml class diagram figure 4.19 important issue consider determine problem small solve directly long require create additional task sumtask occur number element sum value threshold figure 4.18 arbitrarily set 1,000 practice determine problem solve directly require careful timing trial value vary accord implementation interesting java fork join model management task library construct pool worker thread balance load task available worker situation thousand task handful thread perform work example separate thread cpu additionally thread forkjoinpool maintain queue task fork thread queue steal task thread queue work steal algorithm balance workload task thread public class sumtask extend recursivetask < integer > static final int threshold = 1000 private int begin private int end private int array public sumtask(int begin int end int array this.begin = begin this.end = end this.array = array protected integer compute end begin < threshold int sum = 0 int = begin < = end i++ sum + = array[i int mid = begin + end 2 sumtask lefttask = new sumtask(begin mid array sumtask righttask = new sumtask(mid + 1 end array return righttask.join + lefttask.join fork

join calculation java api openmp set compiler directive api program write c c++ fortran provide support parallel programming shared- memory environment openmp identify parallel region block code run parallel application developer insert compiler directive code parallel region directive instruct openmp run- threads concurrency uml class diagram java fork join time library execute region parallel follow c program illus- trate compiler directive parallel region contain printf int main(int argc char argv sequential code pragma omp parallel printf("i parallel region sequential code openmp encounter directive pragma omp parallel create thread processing core system dual core system thread create quad core system create forth thread simultaneously execute parallel region thread exit parallel region terminate openmp provide additional directive run code region parallel include parallelize loop example assume array b size n. wish sum content place result array c. task run parallel following code segment contain compiler directive parallelize loop pragma omp parallel = 0 < n i++ c[i = a[i + b[i openmp divide work contain loop thread create response directive pragma omp parallel addition provide directive parallelization openmp allow developer choose level parallelism example set number thread manually allow developer identify datum share thread private thread

openmp available open source commercial compiler linux win- dows macos system encourage reader interested learn openmp consult bibliography end chapter grand central dispatch grand central dispatch gcd technology develop apple macos ios operate system combination run time library api language extension allow developer identify section code task run parallel like openmp gcd manage detail gcd schedule task run time execution place dispatch queue remove task queue assign task available thread pool thread manage gcd identify type dispatch queue serial concurrent task place serial queue remove fifo order task remove queue complete execution task remove process serial queue know main queue developer create additional serial queue local particular process serial queue know private dispatch queue serial queue useful ensure sequential execution task task place concurrent queue remove fifo order task remove time allow multiple task execute parallel system wide concurrent queue know global dispatch queue divide primary quality service qos class user interactive user interactive class represent task interact user user interface event handling ensure responsive user interface complete task belong class require small work qos class user initiate user initiate class similar user interactive class task associate responsive user interface user initiate task require long process threads concurrency time open file url user initiate task example task belong class complete user continue inter- act system need service quickly task user interactive queue qos class utility utility class represent task require long time complete demand immediate result class include work import datum qos class background task belong background class visible user time sensitive example include index- e mailbox system perform backup task submit dispatch queue express 1 c c++ objective c language gcd identify language extension know block simply self contain unit work block specify caret ˆ insert pair brace code brace identify unit work perform simple example block show ^

brace identify unit work perform simple example block show ^ printf("i block 2 swift programming language task define closure similar block express self contain unit functionality syntactically swift closure write way block minus lead caret follow swift code segment illustrate obtain concurrent queue user initiate class submit task queue dispatch async function let queue = dispatch global queue qos class user initiate 0 dispatch async(queue print("i closure internally gcd thread pool compose posix thread gcd actively manage pool allow number thread grow shrink accord- e application demand system capacity gcd implement libdispatch library apple release apache commons license port freebsd operate system intel thread building blocks intel threading building block tbb template library support design- e parallel application c++ library require special compiler language support developer specify task run par- allel tbb task scheduler map task underlie thread furthermore task scheduler provide load balancing cache aware mean precedence task likely datum store cache memory execute quickly tbb provide rich set feature include template parallel loop structure atomic operation mutual exclusion locking addition provide concurrent datum struc- ture include hash map queue vector serve equivalent thread safe version c++ standard template library datum structure let use parallel loop example initially assume func- tion name apply(float value perform operation parameter value array v size n contain float value use following serial loop pass value v apply function int = 0 < n i++ developer manually apply datum parallelism section 4.2.2 multicore system assign different region array v pro- cesse core tie technique achieve parallelism closely physical hardware algorithm modify recompile number process core specific architecture alternatively developer use tbb provide parallel template expect value parallel range range refer range element iterate know iteration space body specify operation perform subrange element rewrite serial loop tbb parallel template follow parallel size t(0 n [=

loop tbb parallel template follow parallel size t(0 n [= size t apply(v[i parameter specify iteration space 0 n1 correspond number element array v second parameter c++ lambda function require bit explanation expression [= size t parameter assume value iteration space case 0 1 value identify array element v pass parameter apply(v[i tbb library divide loop iteration separate chunk create number task operate chunk parallel function allow developer manually specify size chunk wish tbb create number thread assign task available thread similar fork join library java advantage approach require developer identify operation run parallel specify parallel loop library man- thread concurrency age detail involve divide work separate task run parallel intel tbb commercial open source version run windows linux macos refer bibliography detail develop parallel application tbb section discuss issue consider design multi- fork exec system call chapter 3 describe fork system create separate duplicate process semantic fork exec system call change multithreaded program thread program call fork new process duplicate thread new process single threaded unix system choose version fork duplicate thread duplicate thread invoke fork system exec system typically work way describe chapter 3 thread invoke exec system program specify parameter exec replace entire process include version fork use depend application exec call immediately forking duplicate thread unnecessary program specify parameter exec replace process instance duplicate call thread appropri- eat separate process exec fork separate process duplicate thread signal unix system notify process particular event occur signal receive synchronously asynchronously depend source reason event signal signal synchronous asynchronous follow pattern 1 signal generate occurrence particular event 2 signal deliver process 3 deliver signal handle example synchronous signal include illegal memory access divi- sion 0 run program perform action signal gen- erate synchronous signal deliver process perform operation cause signal reason consider signal generate event external run process process

signal reason consider signal generate event external run process process receive signal asynchronously example signal include terminate process specific keystroke < control><c > have timer expire typically asynchronous signal send signal handle possible handler 1 default signal handler 2 user define signal handler signal default signal handler kernel run han- dle signal default action override user define signal handler call handle signal signal handle differ- ent way signal ignore example illegal memory access handle terminate program handle signal single threaded program straightforward signal deliver process deliver signal compli- cat multithreaded program process thread signal deliver general following option exist 1 deliver signal thread signal apply 2 deliver signal thread process 3 deliver signal certain thread process 4 assign specific thread receive signal process method deliver signal depend type signal generate example synchronous signal need deliver thread cause signal thread process situation asynchronous signal clear asynchronous signal signal terminate process < control><c > example)—should send thread standard unix function deliver signal kill(pid t pid int signal function specify process pid particular signal signal deliver multithreaded version unix allow thread specify signal accept block case asynchronous signal deliver thread block signal need handle signal typically deliver thread find block posix pthreads provide following function allow signal deliver specify thread tid pthread kill(pthread t tid int signal windows explicitly provide support signal allow emulate asynchronous procedure call apcs apc facility enable user thread specify function call user thread receive notification particular event indicate threads concurrency apc roughly equivalent asynchronous signal unix unix contend deal signal mul- tithreade environment apc facility straightforward apc deliver particular thread process thread cancellation involve terminate thread complete example multiple thread concurrently search database thread return result remain thread cancel situation occur user press button web browser stop web page load web page load thread image load separate thread

page load web page load thread image load separate thread user press stop button browser thread load page thread cancel refer target thread cancellation target thread occur different scenario 1 asynchronous cancellation thread immediately terminate tar- 2 defer cancellation target thread periodically check terminate allow opportunity terminate difficulty cancellation occur situation resource allocate cancel thread thread cancel midst update datum share thread especially troublesome asynchronous cancellation operate system reclaim system resource cancel thread reclaim resource cancel thread asynchronously free necessary system wide resource defer cancellation contrast thread indicate target thread cancel cancellation occur target thread check flag determine cancel thread perform check point cancel safely pthreads thread cancellation initiate pthread cancel function identifier target thread pass parameter func- tion

following code illustrate create cancel thread pthread t tid create thread pthread create(&tid 0 worker null cancel thread wait thread terminate invoke pthread cancel()indicate request cancel target thread actual cancellation depend target thread set handle request target thread finally cancel pthread join cancel thread return pthreads support cancellation mode mode define state type illustrate table athread set cancellation state type api table illustrate pthreads allow thread disable enable can- cellation obviously thread cancel cancellation disable cancellation request remain pende thread later enable cancellation respond request default cancellation type defer cancellation cancella- tion occur thread reach cancellation point block- e system call posix standard c library define cancellation point list invoke command man pthread linux system example read system cancellation point allow cancel thread block await input read technique establish cancellation point invoke pthread testcancel function cancellation request find pende

pthread testcancel return thread terminate function return thread continue run additionally pthreads allow function know cleanup handler invoke thread cancel function allow resource thread acquire release thread follow code illustrate thread respond cancellation request defer cancellation 1 work awhile check cancellation request issue describe early asynchronous cancellation recommend pthreads documentation cover interesting note linux system thread cancellation pthreads api handle signal section 4.6.2 thread cancellation java use policy similar defer cancellation pthreads cancel java thread invoke interrupt method set interruption status target thread true thread concurrency set interruption status thread thread check interruption status invoke isinter- rupted method return boolean value thread interruption thread.currentthread().isinterrupted thread belong process share datum process data sharing provide benefit multithreaded programming circumstance thread need copy certain datum datum thread local storage tls example transaction process system service transaction separate thread furthermore transaction assign unique identifier associate thread unique transaction identifier use thread local storage easy confuse tls local variable local variable visible single function invocation tls datum visible function invocation additionally developer control thread creation process example implicit technique thread pool alternative approach necessary way tls similar static datum difference tls datum unique thread fact tls usually declare static thread library compiler provide support tls example java provide threadlocal < t > class set method threadlocal < t > object pthread include type pthread key t provide key specific thread key access tls datum microsoft c language simply require add storage attribute threadstatic declare thread local datum gcc compiler provide storage class keyword thread declare tls datum example wish assign unique identifier thread declare thread int threadid final issue consider multithreaded program concern commu- nication kernel thread library require level model discuss section 4.3.3 coordination allow number kernel thread dynamically adjust help ensure good performance system implement level model place intermediate data structure user kernel

system implement level model place intermediate data structure user kernel thread data structure typically know lightweight process lwp show figure 4.20 user thread library lwp appear virtual processor application schedule user thread run lwp attach kernel thread kernel thread operate system schedule run physical processor kernel thread block wait o operation complete lwp block chain user level thread attach lwp block application require number lwps run efficiently consider cpu bind application run single processor scenario thread run time lwp sufficient application o- intensive require multiple lwp execute typically lwp require concurrent block system suppose example different file read request occur simultaneously lwp need wait o completion kernel process lwp fifth request wait lwps return kernel scheme communication user thread library kernel know scheduler activation work follow kernel pro- vide application set virtual processor lwps application schedule user thread available virtual processor furthermore kernel inform application certain event procedure know upcall upcall handle thread library upcall handler upcall handler run virtual processor event trigger upcall occur application thread block scenario kernel make upcall application inform thread block identify specific thread kernel allocate new virtual processor application appli- cation run upcall handler new virtual processor save lightweight process lwp thread concurrency state block thread relinquish virtual processor block thread run upcall handler schedule thread eligible run new virtual processor event block thread wait occur kernel make upcall thread library inform previously block thread eligible run upcall handler event require virtual processor kernel allocate new virtual processor preempt user thread run upcall handler virtual processor mark unblocked thread eligible run application schedule eligible thread run available virtual processor point examine number concept issue relate thread conclude chapter explore thread implement windows linux system windows application run separate process process contain thread windows api create thread cover section 4.4.2 additionally windows use mapping describe

create thread cover section 4.4.2 additionally windows use mapping describe section 4.3.2 user level thread map associate kernel general component thread include thread id uniquely identify thread register set represent status processor program counter user stack employ thread run user mode kernel stack employ thread run kernel mode aprivate storage area run time library dynamic link register set stack private storage area know context primary data structure thread include ethread executive thread block kthread kernel thread block teb thread environment block key component ethread include pointer process thread belong address routine thread start control ethread contain pointer corresponding data structure windows thread kthread include scheduling synchronization information thread addition kthread include kernel stack thread run kernel mode pointer teb ethread kthread exist entirely kernel space mean kernel access teb user space data structure access thread run user mode field teb contain thread identifier user mode stack array thread local storage structure windows thread illustrate figure linux provide fork system traditional functionality duplicate process describe chapter 3 linux provide ability create thread clone system linux distinguish process thread fact linux use term task process thread refer flow control clone invoke pass set flag determine sharing place parent child task flag list figure 4.22 example suppose clone pass flag clone fs clone vm clone sighand clone file parent child task share file system information current working directory memory space signal handler threads concurrency file system information share memory space share signal handler share set open file share flag pass clone invoke set open file clone fashion equivalent create thread describe chapter parent task share resource child task flag set clone invoke sharing take place result functionality similar provide fork system vary level sharing possible way task repre- sente linux kernel unique kernel datum structure specifically struct task struct exist task system data datum structure specifically struct task struct exist task system data structure instead store datum task contain pointer data structure datum store example datum structure represent list open file signal handling information virtual memory fork invoke new task create copy associate datum structure parent process new task create clone system copy datum structure new task point data structure parent task depend set flag pass clone finally flexibility clone system extend concept container virtualization topic introduce chapter 1 recall chapter container virtualization technique pro- vide operate system allow create multiple linux system container single linux kernel run isolation certain flag pass clone distinguish create task behave like process thread base sharing parent child task flag pass clone allow linux container create container cover fully chapter 18 thread represent basic unit cpu utilization thread belong process share process resource include code primary benefit multithreaded application 1 respon- siveness 2 resource sharing 3 economy 4 scalability concurrency exist multiple thread make progress parallelism exist multiple thread make progress simulta- neously system single cpu concurrency possible parallelism require multicore system provide multiple cpu challenge design multithreaded application include divide balance work divide datum different thread identify datum dependency finally mul- tithreade program especially challenging test debug data parallelism distribute subset datum different com- puting core perform operation core task paral- lelism distribute datum task multiple core task run unique operation user application create user level thread ultimately map kernel thread execute cpu model map user level thread kernel thread approach include model thread library provide api create manage thread common thread library include windows pthreads java threading windows windows system pthreads available posix compatible system unix linux macos java thread run system support java virtual machine implicit threading involve identify task thread allow language api framework create manage thread approach implicit threading include thread pool framework create manage thread approach implicit threading include thread pool fork join framework grand central dispatch implicit threading increasingly common technique programmer use develop concurrent parallel application thread terminate asynchronous deferred cancel- lation asynchronous cancellation stop thread immediately middle perform update deferred cancellation inform thread terminate allow thread terminate orderly fashion circumstance deferred cancellation prefer asynchronous termination unlike operate system linux distinguish process thread instead refer task linux clone system create task behave like process like thread provide programming example multithreading provide well performance single threaded solution amdahl law calculate speedup gain application 60 percent parallel component processing core b processing core thread concurrency multithreaded web server describe section 4.1 exhibit task datum parallelism difference user level thread kernel level thread circumstance type well describe action take kernel context switch kernel- resource thread create differ process create assume operate system map user level thread kernel model mapping lwp furthermore system allow developer create real time thread use real time system necessary bind real time thread lwp explain vahalia 1996 cover threading version unix mcdougall mauro 2007 describe development thread solaris kernel russi- novich et al 2017 discuss threading windows operate system family mauerer 2008 love 2010 explain linux handle threading levin 2013 cover thread macos ios herlihy shavit 2012 cover parallelism issue multicore system aubanel 2017 cover paral- lelism different algorithm e. aubanel elements parallel computing crc press 2017 herlihy shavit 2012 m. herlihy n. shavit art multiprocessor programming revised edition morgan kaufmann publishers inc. 2012 j. levin mac os x ios internals apple core

wiley r. love linux kernel development edition developer w. mauerer professional linux kernel architecture john wiley sons 2008 mcdougall mauro 2007 r. mcdougall j. mauro solaris internals second edition prentice hall 2007 russinovich et al 2017 m. russinovich d. a. solomon a. ionescu win- dows internals 1 seventh edition microsoft press 2017 u. vahalia unix internals new frontiers prentice hall chapter 4 exercise provide programming example multithreading provide well performance single threaded solution circumstance multithreaded solution multi- ple kernel thread provide well performance single threaded solution single processor system following component program state share thread multithreaded process multithreaded solution multiple user level thread achieve well performance multiprocessor system single processor system explain chapter 3 discuss google chrome browser practice open new tab separate process benefit achieve instead chrome design open new tab separate thread explain possible concurrency parallelism explain amdahl law calculate speedup gain following appli- 40 percent parallel processing core b sixteen 67 percent parallel processing core b pro- 90 percent parallel processing core b pro- determine following problem exhibit task datum parallelism separate thread generate thumbnail photo transpose matrix parallel networked application thread read network write network fork join array summation application describe section 4.5.2 grand central dispatch system system dual core processor processor available scheduling cpu intensive application run system input perform program start single file open similarly output perform program termi- nate program result write single file start termination program entirely cpu bind task improve performance application multithreade application run system use threading model user thread map kernel thread thread create perform input output thread create cpu intensive portion application explain consider following code segment pid t pid pid = fork pid = = 0

child process thread create unique process create unique thread create describe section 4.7.2 linux distinguish pro- cesse thread instead linux treat way allow task akin process thread depend set flag pass clone system operate system windows treat process thread differently typically system use notation data structure process contain pointer separate thread belong process contrast approach model process thread kernel program show figure 4.23 use pthreads api output program line c line p consider multicore system multithreaded program write threading model let number user level thread program great number processing core system discuss performance implication following number kernel thread allocate program number processing core number kernel thread allocate program equal number processing core number kernel thread allocate program great number processing core number int value = 0 void runner(void param thread int main(int argc char argv pid t pid pthread t tid pthread attr t attr pid = fork pid = = 0 child process pthread attr init(&attr printf("child value = d",value line c pid > 0 parent process printf("parent value = d",value line p void runner(void param value = 5 c program exercise 4.19

pthread provide api manage thread cancellation pthread setcancelstate function set cancellation state prototype appear follow pthread setcancelstate(int state int oldstate possible value state pthread cancel enable pthread cancel disable code segment show figure 4.24 provide example operation suitable perform call disable enable thread cancellation pthread setcancelstate(pthread cancel disable oldstate operation perform pthread setcancelstate(pthread cancel enable oldstate c program exercise 4.21 write multithreaded program calculate statistical value list number program pass series number command line create separate worker thread thread determine average number second determine maximum value determine mini- mum value example suppose program pass integer 90 81 78 95 79 72 85 program report average value 82 minimum value 72 maximum value 95 variable represent average minimum maximum value store globally worker thread set value parent thread output value worker exit obviously expand program create additional thread determine statistical value median standard write multithreaded program output prime number pro- gram work follow user run program enter number command line program create separate thread output prime number equal number enter user interesting way calculate π use technique know monte carlo involve randomization technique work follow suppose circle inscribe square show figure 4.25 assume radius circle 1 generate series random point simple x y coordinate point fall cartesian coordinate bind square total number random point generate occur circle estimate π perform following calculation π = 4× number point circle total number point write multithreaded version algorithm create separate thread generate number random point thread count number point occur circle store result global variable thread exit parent thread calculate output estimate value π worth experiment number random point generate general rule great number point close approximation π threads concurrency monte carlo technique calculate π source code download text find sample program provide technique generate random number determine random x y point occur circle

generate random number determine random x y point occur circle reader interested detail monte carlo method estimate π consult bibliography end chapter chapter 6 modify exercise relevant material repeat exercise 4.24 instead separate thread generate random point use openmp parallelize generation point careful place calculation π parallel region want calculate π

modify socket base date server figure 3.27 chapter 3 server service client request separate thread fibonacci sequence series number 0 1 1 2 3 5 8

for- mally express fib0 = 0 fib1 = 1 fibn = fibn1 + fibn2 write multithreaded program generate fibonacci sequence program work follow command line user enter number fibonacci number program gen- erate program create separate thread generate fibonacci number place sequence datum share thread array probably convenient datum struc- ture thread finish execution parent thread output sequence generate child thread parent thread begin output fibonacci sequence child thread finish parent thread wait child thread finish use technique describe section 4.4 meet requirement modify programming problem exercise 3.20 chapter 3 ask design pid manager modification consist write multithreaded program test solution exercise 3.20 create number thread example 100 thread request pid sleep random period time release pid sleep random period time approximate typical pid usage pid assign new

process process execute terminate pid release process termina- tion unix linux system sleeping accomplish sleep function pass integer value represent number second sleep problem modify chapter 7 exercise 3.25 chapter 3 involve design echo server java threading api server single threaded mean server respond concurrent echo client current client exit modify solution exercise 3.25 echo server service client separate request solution 9 × 9 sudoku puzzle project 1 sudoku solution validator sudoku puzzle use 9 × 9 grid column row 3 × 3 subgrid contain digit 1 9 figure 4.26 present example valid sudoku puzzle project consist
design multithreaded application determine solution sudoku puzzle valid different way multithreade application suggest strategy create thread check following criterion thread check column contain digit 1 9 thread check row contain digit 1 9 thread concurrency thread check 3 × 3 subgrid contain digit 1 result total separate thread validate sudoku puzzle welcome create thread project example create thread check column create separate thread check i. pass parameter thread parent thread create worker thread pass worker

pass parameter thread parent thread create worker thread pass worker location check sudoku grid step require pass parameter thread easy approach create data structure struct example structure pass row column thread begin validate appear follow structure pass datum thread pthreads windows program create worker thread strategy similar show parameter datum = parameter malloc(sizeof(parameters data->row = 1 data->column = 1 create thread pass datum parameter datum pointer pass pthread create pthreads function createthread windows function turn pass parameter function run separate thread ii return result parent thread worker thread assign task determine validity partic- ular region sudoku puzzle worker perform check pass result parent good way handle create array integer value visible thread ith index array correspond ith worker thread worker set corresponding value 1

indicate region sudoku puzzle valid value 0 indicate worker thread complete parent thread check entry result array determine sudoku puzzle project 2 multithreaded sorting application write multithreaded sorting program work follow list integer divide small list equal size separate thread term sort thread sort sublist sort algorithm choice sublist merge thread merge thread merge sublist single sort list global datum share thread easy way set datum create global array sort thread work half array second global array size unsorted integer array establish merge thread merge sublist second array graphically program structure figure 4.27 programming project

require pass parameter sort thread particular necessary identify start index thread begin sort refer instruction project 1 detail pass parameter thread parent thread output sort array sort thread 7 12 19 3 18 7 12 19 3 18 4 2 6 15 8 2 3 4 6 7 8 12 15 18 19 4 2 6 15 8 project 3 fork join sorting application implement precede project multithreaded sorting application java fork join parallelism api project develop different version version implement different divide conquer sorting quicksort implementation use quicksort algorithm divide list element sort left half right half base thread concurrency position pivot value mergesort algorithm divide list evenly sized half quicksort mergesort algorithm list sort fall threshold value example list size 100 few directly apply simple algorithm selection insertion sort data structure text describe know divide conquer sorting algorithm class sumtask show section 4.5.2.1 extend recursivetask result bear forkjointask assignment involve sort array pass task return value instead create class extend recursiveaction non result bear forkjointask figure 4.19 object pass sort algorithm require implement java comparable interface need reflect class definition sort algorithm source code download text include java code provide foundation begin project c h p t e r cpu scheduling basis multiprogrammed operate system switch- e cpu process operate system computer productive chapter introduce basic cpu scheduling concept present cpu scheduling algorithm include real time system consider problem select algorithm particular system chapter 4 introduce thread process model modern oper- ating system kernel level thread process fact schedule operate system term process scheduling thread scheduling interchangeably chapter use process scheduling discuss general scheduling concept thread scheduling refer thread specific idea similarly chapter 1 describe core basic computational unit cpu process execute cpu core instance chapter use general terminology schedule process run cpu imply process run describe cpu

terminology schedule process run cpu imply process run describe cpu scheduling algorithm assess cpu scheduling algorithm base scheduling criterion explain issue relate multiprocessor multicore scheduling describe real time scheduling algorithm describe scheduling algorithm windows linux solaris operating system apply modeling simulation evaluate cpu scheduling algorithm design program implement different cpu scheduling algo- system single cpu core process run time wait cpu core free reschedule objective multiprogramming process run time maximize cpu utilization idea relatively simple aprocess execute wait typically completion o request simple computer system cpu sit idle waiting time waste useful work accomplish multiprogramming try use time produc- tively process keep memory time process wait operate system take cpu away process give cpu process pattern continue time process wait process use cpu multicore system concept keep cpu busy extend processing core scheduling kind fundamental operating system function computer resource schedule use cpu course primary computer resource scheduling central read file write file read file wait o wait o wait o alternate sequence cpu o burst cpu o burst cycle success cpu scheduling depend observe property process process execution consist cycle cpu execution o wait process alternate state process execution begin cpu burst follow o burst follow cpu burst o burst eventually final cpu burst end system request terminate execution figure 5.1 duration cpu burst measure extensively vary greatly process process computer computer tend frequency curve similar show figure 5.2 curve generally characterize exponential hyperexponential large number short cpu burst small number long cpu burst o bind program typically short cpu burst cpu bind program long cpu burst distribution important implement cpu scheduling algorithm cpu idle operate system select process ready queue execute selection process carry cpu scheduler select process process memory ready execute allocate cpu process note ready queue necessarily fifo queue shall consider scheduling algorithm ready queue implement

necessarily fifo queue shall consider scheduling algorithm ready queue implement fifo queue priority queue tree simply unordered link list conceptually process ready queue line wait chance run cpu record queue generally process control block pcb process histogram cpu burst duration preemptive nonpreemptive scheduling cpu scheduling decision place follow circum- 1 process switch running state waiting state example result o request invocation wait termination child process 2 process switch running state ready state example interrupt occur 3 process switch waiting state ready state example completion o 4 process terminate situation 1 4 choice term scheduling new process exist ready queue select execution choice situation 2 3 scheduling take place circumstance 1 4 scheduling scheme nonpreemptive cooperative pre- emptive nonpreemptive scheduling cpu allocate process process keep cpu release terminate switch waiting state virtually modern operate system include windows macos linux unix use preemptive scheduling algo- unfortunately preemptive scheduling result race condition datum share process consider case process share datum process update datum preempt second process run second process try read datum inconsistent state issue explore detail chapter 6 preemption affect design operate system kernel processing system kernel busy activity behalf process activity involve change important kernel datum instance o queue happen process preempt middle change kernel device driver need read modify structure chaos ensue discuss section 6.2 operate system kernel design nonpreemptive preemptive nonpreemptive kernel wait system complete process block wait o complete place kernel preempt process kernel datum structure inconsistent state unfortunately kernel execution model poor support real time computing task complete execution give time frame section 5.6 explore scheduling demand real time system preemptive kernel require mechanism mutex lock prevent race condition access share kernel datum structure modern operate system fully preemptive run role dispatcher interrupt definition occur time ignore kernel section code affect inter- rupt guard simultaneous use operate kernel section code affect inter- rupt guard simultaneous use operate system need accept interrupt time input lose output overwrite section code access concurrently process disable interrupt entry reenable interrupt exit important note section code disable interrupt occur typically contain instruction component involve cpu scheduling function dispatcher dispatcher module give control cpu core process select cpu scheduler function involve following switch context process switch user mode jump proper location user program resume program dispatcher fast possible invoke context switch time take dispatcher stop process start run know dispatch latency illustrate figure interesting question consider context switch occur system wide level number context switch obtain vmstat command available linux system output trim command vmstat 1 3 command provide 3 line output 1 second delay line give average number context switch 1 second system boot line number context switch 1 second interval machine boot aver- age 24 context switch second past second 225 context switch 339 context switch second prior use /proc file system determine number context switch give process example content file /proc/2166 status list statistic process pid =

2166 command provide following trim output voluntary ctxt switch nonvoluntary ctxt switch output show number context switch lifetime process notice distinction voluntary nonvoluntary context switch voluntary context switch occur process give control cpu require resource currently unavailable blocking o. anonvoluntary context switch occur cpu take away process time slice expire preempt high priority process different cpu scheduling algorithm different property choice particular algorithm favor class process choose algorithm use particular situation consider property algorithm criterion suggest compare cpu scheduling algo- rithm characteristic comparison substantial difference algorithm judge good criterion include cpu utilization want cpu busy possible concep- tually cpu utilization range 0 100 percent real system range 40 percent lightly load system 90 percent heavily load system cpu utilization obtain command linux macos unix system throughput cpu busy execute process work measure work number process complete time unit call throughput long process rate process second short transaction ten process second turnaround time point view particular process important criterion long take execute process interval time submission process time completion turnaround time turnaround time sum period spend wait waiting time cpu scheduling algorithm affect time process execute o. affect time process spend wait ready queue waiting time sum period spend wait ready queue response time interactive system turnaround time good criterion process

produce output fairly early continue compute new result previous result output user measure time submission request response produce measure call response time time take start respond time take output response desirable maximize cpu utilization throughput minimize turnaround time waiting time response time case optimize average measure circumstance prefer opti- mize minimum maximum value average example guarantee user good service want minimize maximum response time investigator suggest interactive system pc desktop laptop system important minimize variance response time minimize average response time system reasonable predictable response time consider desirable system fast average highly variable little work cpu scheduling algorithm minimize variance discuss cpu scheduling algorithm follow section illustrate operation accurate illustration involve process sequence cpu burst o burst simplicity consider cpu burst millisecond process example measure comparison average waiting time elaborate evaluation mechanism discuss section 5.8 cpu scheduling deal problem decide process ready queue allocate cpu core different cpu-scheduling algorithm section describe modern cpu architecture multiple processing core describe scheduling algorithm context processing core avail- able single cpu single processing core system capable run process time section 5.5 discuss cpu scheduling context multiprocessor system come serve scheduling far simple cpu scheduling algorithm come serve fcfs scheduling algorithm scheme process request cpu allocate cpu implementation fcfs policy easily manage fifo queue process enter ready queue pcb link tail queue cpu free allocate process head queue run process remove queue code fcfs scheduling simple write understand negative average waiting time fcfs policy long consider following set process arrive time 0 length cpu burst give millisecond process arrive order p1 p2 p3 serve fcfs order result show follow gantt chart bar chart illustrate particular schedule include start finish time participate process waiting time 0 millisecond process p1 24 millisecond process p2 27 millisecond process p3 average waiting time 0 + 24 + 27)/3

process p3 average waiting time 0 + 24 + 27)/3 = 17 millisecond process arrive order p2 p3 p1 result show follow gantt chart average waiting time 6 + 0 + 3)/3 = 3 millisecond reduction substantial average waiting time fcfs policy generally minimal vary substantially process cpu burst time vary addition consider performance fcfs scheduling dynamic situation assume cpu bind process o bind pro- cesse process flow system following scenario result cpu bind process hold cpu time process finish o ready queue wait cpu process wait ready queue o device idle eventually cpu bind process finish cpu burst move o device o bind process short cpu burst execute quickly o queue point cpu sit idle cpu bind process ready queue allocate cpu o process end wait ready queue cpu bind process convoy effect process wait big process cpu effect result low cpu device utilization possible short process allow note fcfs scheduling algorithm nonpreemptive cpu allocate process process keep cpu release cpu terminate request o.

fcfs algorithm particularly troublesome interactive system important process share cpu regular interval disastrous allow process cpu extended period different approach cpu scheduling short job fir sjf schedul- e algorithm algorithm associate process length process cpu burst cpu available assign process small cpu burst cpu burst process fcfs scheduling break tie note appro- priate term scheduling method short cpu burst algorithm scheduling depend length cpu burst process total length use term sjf people textbook use term refer type scheduling example sjf scheduling consider following set process length cpu burst give millisecond sjf scheduling schedule process accord follow gantt chart waiting time 3

millisecond process p1 16 millisecond process p2 9 millisecond process p3 0 millisecond process p4 average waiting time 3 + 16 + 9 + 0)/4 = 7 millisecond comparison fcfs scheduling scheme average waiting time 10.25 millisecond sjf scheduling algorithm provably optimal give mini- mum average waiting time give set process move short process long decrease waiting time short process increase waiting time long process consequently average waiting time decrease sjf algorithm optimal implement level cpu scheduling way know length cpu burst approach problem try approximate sjf scheduling know length cpu burst able predict value expect cpu burst similar length previous one compute approximation length cpu burst pick process short predict cpu burst cpu burst generally predict exponential average measure length previous cpu burst define exponential average follow formula let tn length nth cpu burst let τn+1 predict value cpu burst α 0 α τn+1 = α tn + 1 α)τn value tn contain recent
information τn store past history parameter α control relative weight recent past history prediction α = 0 τn+1 = τn recent history effect current condition assume transient α = 1 τn+1 = tn

recent cpu burst matter history assume old irrelevant commonly α = 1/2 recent history past history equally weight initial τ0 define constant overall system average figure 5.4 show exponential average α = 1/2 τ0 = 10 cpu burst ti prediction length cpu burst understand behavior exponential average expand formula τn+1 substitute τn find τn+1 = αtn + 1 α)αtn1
+ + 1 α)jαtnj + + 1 α)n+1τ0 typically α 1 result 1 α 1 successive term weight predecessor sjf
algorithm preemptive nonpreemptive choice arise new process arrive ready queue previous pro- cess execute cpu burst newly arrive process short leave currently execute process preemptive sjf algorithm preempt currently execute process non- preemptive sjf algorithm allow currently run process finish cpu burst preemptive sjf scheduling call short remaining- example

consider follow process length cpu burst give millisecond process arrive ready queue time

show need indicate burst time result preemptive sjf schedule depict follow gantt chart process p1 start time 0 process queue process p2 arrive time 1 remain time process p1 7 millisecond large time require process p2 4 millisecond process p1 preempt process p2 schedule average waiting time example 10 1 + 1 1 + 17 2 + 5 3)]/4 = 26/4 = 6.5 millisecond nonpreemptive
sjf scheduling result average waiting time 7.75 round robin rr scheduling algorithm similar fcfs scheduling preemption add enable system switch process asmall unit time call time quantum time slice define time quantum generally 10 100 millisecond length ready queue treat circular queue cpu scheduler go ready queue allocate cpu process time interval 1 time quantum implement rr scheduling treat ready queue fifo queue process new process add tail ready queue cpu scheduler pick process ready queue set timer interrupt 1 time quantum dispatch process thing happen process cpu burst 1 time quantum case process release cpu voluntarily scheduler proceed process ready queue cpu burst currently run process long 1 time quantum timer cause interrupt operate system context switch execute process tail ready queue cpu scheduler select process ready queue average waiting time rr policy long consider follow set process arrive time 0 length cpu burst give millisecond use time quantum 4 millisecond process p1 get 4 millisecond require 20 millisecond preempt time quantum cpu give process queue process p2 process p2 need 4 millisecond quit time quantum expire cpu give process process p3 process receive 1 time quantum cpu return process p1 additional time quantum result rr schedule follow let calculate average waiting time schedule p1 wait 6 mil- lisecond 10 4 p2 wait 4 millisecond p3 wait 7 millisecond average waiting

p2 wait 4 millisecond p3 wait 7 millisecond average waiting time 17/3 = 5.66 millisecond

rr scheduling algorithm process allocate cpu 1 time quantum row runnable process process cpu burst exceed 1 time quantum process preempt ready queue rr scheduling algorithm preemptive n process ready queue time quantum q process get 1 n cpu time chunk q time unit process wait long n 1 × q time unit time quan- tum example process time quantum 20 millisecond process 20 millisecond 100 millisecond performance rr algorithm depend heavily size time quantum extreme time quantum extremely large rr policy fcfs policy contrast time quantum extremely small 1 millisecond rr approach result large process time = 10 small time quantum increase context switch number context switch assume example process 10 time unit quantum 12 time unit process finish 1 time quantum overhead quantum 6 time unit process require 2 quantum result context switch time quantum 1 time unit context switch occur slow execution process accordingly figure 5.5 want time quantum large respect context-switch time context switch time approximately 10 percent time quantum 10 percent cpu time spend context switching practice modern system time quantum range 10 100 millisecond time require context switch typically 10 microsecond context switch time small fraction turnaround time depend size time quantum figure 5.6 average turnaround time set process necessarily improve time quantum size increase general average turnaround time improve process finish cpu burst single time quantum example give process 10 time unit quantum 1 time unit average turnaround time 29 time quantum 10 average turnaround time drop 20 context switch time add average turnaround time increase small time quantum context switch time quantum large compare context- switch time large point early time quantum large rr scheduling degenerate fcfs policy rule thumb 80 percent cpu burst short time sjf algorithm special case general priority scheduling algorithm priority associate process cpu allocate average turnaround

priority scheduling algorithm priority associate process cpu allocate average turnaround time turnaround time vary time quantum process high priority equal priority process schedule fcfs order sjf algorithm simply priority algorithm priority p inverse predict cpu burst large cpu burst low priority vice versa note discuss scheduling term high priority low priority priority

generally indicate fix range number 0 7 0 4,095 general agreement 0 high low priority system use low number represent low priority use low number high priority difference lead confusion text assume low number represent high priority example consider following set process assume arrive time 0 order p1 p2 p5 length cpu burst give millisecond priority scheduling schedule process accord follow gantt chart average waiting time 8.2 millisecond priority define internally externally internally define priority use measurable quantity quantity compute priority process example time limit memory requirement number open file ratio average o burst average cpu burst compute priority external priority set criterion outside operate system importance process type fund pay computer use department sponsor work political factor priority scheduling preemptive nonpreemptive process arrive ready queue priority compare priority currently run process apreemptive priority scheduling algorithm preempt cpu priority newly arrive process high priority currently run process anonpreemptive priority scheduling algorithm simply new process head ready queue major problem priority scheduling algorithm indefinit block- ing starvation process ready run wait cpu consider block priority scheduling algorithm leave low- priority process wait indefinitely heavily load computer system steady stream high priority process prevent low priority process get cpu generally thing happen process eventually run 2 a.m. sunday system finally lightly load computer system eventually crash lose unfin- 7094 mit 1973 find low priority process submit 1967 run asolution problem indefinite blockage low priority process aging aging involve gradually increase priority process wait system long time example priority range 127 low 0 high periodically second increase priority wait process 1 eventually process initial priority 127 high priority system execute fact little 2 minute priority-127 process age option combine round robin priority scheduling way system execute high priority process run process priority round robin scheduling let illustrate example follow set process burst time millisec- priority scheduling round robin process equal priority schedule process accord follow

scheduling round robin process equal priority schedule process accord follow gantt chart time quantum 2 millisecond example process p4 high priority run comple- tion process p2 p3 high priority execute round robin fashion notice process p2 finish time 16 process p3 high priority process run complete execution process p1 p5 remain equal priority execute round robin order complete multilevel queue scheduling priority round robin scheduling process place single queue scheduler select process high priority run depend queue manage o(n search necessary determine high priority process practice easy separate queue distinct priority priority scheduling simply schedule process high priority queue illustrate figure 5.7 approach know multilevel queue work priority scheduling combine round robin multiple process high priority queue execute round robin order generalized form approach priority assign statically process process remain queue duration runtime priority = 2 priority = n priority = 1 priority = 0

separate queue priority multilevel queue scheduling multilevel queue scheduling algorithm partition process separate queue base process type figure 5.8 example common division foreground interac- tive process background batch process type pro- cesse different response time requirement different scheduling need addition foreground process priority exter- nally define background process separate queue foreground background process queue scheduling algorithm foreground queue schedule rr algorithm example background queue schedule fcfs addition scheduling queue com- monly implement fixed priority preemptive scheduling example real time queue absolute priority interactive queue let look example multilevel queue scheduling algorithm queue list order priority 1 real time process 2 system process 3 interactive process 4 batch process queue absolute priority low priority queue process batch queue example run queue real time process system process interactive process interactive process enter ready queue batch process run batch process preempt possibility time slice queue queue get certain portion cpu time schedule var- ious process instance foreground background queue example foreground queue give 80 percent cpu time rr scheduling process background queue receive 20 percent cpu process fcfs basis multilevel feedback queue scheduling normally multilevel queue scheduling algorithm process permanently assign queue enter system separate queue foreground background process example process queue process change foreground background nature setup advantage low scheduling overhead inflexible multilevel feedback queue scheduling algorithm contrast allow process queue idea separate process accord characteristic cpu burst process use cpu time move low priority queue scheme leave o bind interactive process typically characterize short cpu burst high priority queue addition process wait long low priority queue move high priority queue form aging prevent starvation example consider multilevel feedback queue scheduler queue number 0 2 figure 5.9 scheduler execute process queue 0 queue 0 execute process queue 1 similarly process queue 2 execute queue 0 1 process arrive queue 1 preempt process

execute queue 0 1 process arrive queue 1 preempt process queue 2 process queue 1 turn preempt process arrive queue 0 enter process queue 0 process queue 0 give time quantum 8 millisecond finish time move tail queue 1 queue 0 process head queue 1 give quantum 16 millisecond complete preempt queue 2 process queue 2 run fcfs basis run queue 0 1 prevent starvation process wait long low priority queue gradually move high priority quantum = 8 quantum = 16 multilevel feedback queue scheduling algorithm give high priority process cpu burst 8 millisecond process quickly cpu finish cpu burst o burst process need 8 24 millisecond serve quickly low priority short process long process automatically sink queue 2 serve fcfs order cpu cycle leave queue 0

general multilevel feedback queue scheduler define follow- number queue scheduling algorithm queue method determine upgrade process higher- method determine demote process lower- method determine queue process enter process need service definition multilevel feedback queue scheduler make general cpu scheduling algorithm configure match specific system design unfortunately complex algorithm define good scheduler require mean select value parameter chapter 4 introduce thread process model distinguish user level kernel level thread modern operate sys- tem kernel level thread process schedule operate system user level thread manage thread library kernel unaware run cpu user level thread ulti- mately map associate kernel level thread mapping indirect use lightweight process lwp section explore scheduling issue involve user level kernel level thread offer specific example scheduling pthread distinction user level kernel level thread lie schedule system implement section 4.3.1 section 4.3.3 model thread library schedule user- level thread run available lwp scheme know process- contention scope pcs competition cpu take place thread belong process thread library sched- ule user thread available lwp mean thread actually run cpu require operate system schedule lwp kernel thread physical cpu core decide kernel level thread schedule cpu kernel use system contention scope scs competition cpu scs scheduling take place thread system system model section 4.3.2 windows linux schedule thread scs typically pcs accord priority scheduler select runnable thread high priority run user level thread priority set programmer adjust thread library thread library allow programmer change priority thread important note pcs typically preempt thread currently run favor high priority thread guarantee time slicing section 5.3.3 thread equal priority provide sample posix pthread program section 4.4.1 introduction thread creation pthreads highlight posix pthread api allow specify pcs scs thread creation pthreads identify following contention scope value pthread scope process schedule thread pcs scheduling pthread scope system schedule thread scs scheduling pthread scope process

pthread scope system schedule thread scs scheduling pthread scope process policy schedule user level thread available lwp number lwp maintain thread library scheduler activation section 4.6.5 pthread scope system scheduling policy create bind lwp user level thread system effectively map thread policy pthread ipc interprocess communication provide function set get contention scope policy pthread attr setscope(pthread attr t attr int scope pthread attr getscope(pthread attr t attr int scope parameter function contain pointer attribute set thread second parameter pthread attr setscope function pass pthread scope system pthread scope process value indicate contention scope set case pthread attr getscope second parameter contain pointer int value set current value contention scope error occur function return nonzero value figure 5.10

illustrate pthread scheduling api pro- pthread scope system create separate thread run scs scheduling policy note system certain contention scope value allow example linux macos system allow pthread scope system define num thread 5 int main(int argc char argv int scope pthread t tid[num thread pthread attr t attr default attribute pthread attr init(&attr inquire current scope pthread attr getscope(&attr scope = 0 fprintf(stderr unable scheduling scopen scope = = pthread scope process printf("pthread scope process scope = = pthread scope system printf("pthread scope system fprintf(stderr illegal scope value.n set scheduling algorithm pcs scs pthread attr setscope(&attr pthread scope system create thread = 0 < num thread i++ join thread = 0 < num thread i++ pthread join(tid[i null thread begin control function void runner(void param work pthread scheduling api discussion far focus problem schedule cpu system single processing core multiple cpu available load shar- ing multiple thread run parallel possible scheduling issue correspondingly complex possibility

try see cpu scheduling single core cpu good solution traditionally term multiprocessor refer system provide multiple physical processor processor contain single core cpu definition multiprocessor evolve significantly modern compute system multiprocessor apply follow multicore cpu multithreaded core numa system heterogeneous multiprocessing discuss concern multiprocessor scheduling con- text different architecture example concentrate system processor identical homogeneous term functionality use available cpu run process queue example explore system processor identical capability approach multiple processor scheduling approach cpu scheduling multiprocessor system scheduling decision o processing system activity handle single processor master server processor execute user code asymmetric multiprocessing simple core access system datum structure reduce need datum sharing downfall approach master server potential bottleneck overall system performance reduce standard approach support multiprocessor symmetric mul- tiprocessing smp processor self schedule schedule pro- ceed have scheduler processor examine ready queue select thread run note provide possible strategy organize thread eligible schedule 1 thread common ready queue 2 processor private queue thread strategy contrast figure 5.11 select option possible race condition share ready queue ensure separate processor choose schedule thread thread lose queue discuss common ready queue core run queue organization ready queue chapter 6 use form locking protect common ready queue race condition locking highly contend access queue require lock ownership access shared queue likely performance bottleneck second option permit processor schedule thread private run queue suffer possible performance problem associate share run queue common approach system support smp additionally describe section 5.5.4 have private per- processor run queue fact lead efficient use cache memory issue processor run queue notably workload vary size shall balancing algorithm equalize workload processor virtually modern operate system support smp include windows linux macos mobile system include android ios remainder section discuss issue concern smp system design cpu scheduling algorithm traditionally smp system allow process run parallel provide multiple physical

traditionally smp system allow process run parallel provide multiple physical processor contemporary com- puter hardware place multiple compute core physical chip result multicore processor core maintain architectural state appear operate system separate logical cpu smp system use multicore processor fast consume power system cpu physical chip multicore processor complicate scheduling issue let consider happen researcher discover processor access memory spend significant time wait datum available situation know memory stall occur primarily modern processor operate fast speed memory how- memory stall occur cache miss access datum cache memory figure 5.12 illustrate memory stall scenario processor spend 50 percent time wait datum available memory memory stall cycle remedy situation recent hardware design imple- mente multithreaded processing core

hardware thread assign core way hardware thread stall wait memory core switch thread figure 5.13 illus- trate dual threaded processing core execution thread 0 execution thread 1 interleave operate system perspec- tive hardware thread maintain architectural state instruction pointer register set appear logical cpu available run software thread technique know chip multithreading cmt illustrate figure 5.14 processor contain compute core core contain hardware thread perspective operate system logical cpu intel processor use term hyper threading know simultane- ous multithreading smt describe assign multiple hardware thread single processing core contemporary intel processor i7 sup- port thread core oracle sparc m7 processor support thread core core processor provide operate system 64 logical cpu general way multithread processing core coarse- grained fine graine multithreading coarse grained multithread- ing thread execute core long latency event memory stall occur delay cause long latency event core switch thread begin execution cost switch- e thread high instruction pipeline flush thread begin execution processor core new thread begin execution begin fill pipeline instruction fine grained interleave multithreade switch thread fine level granularity typically boundary instruction multithreaded multicore system operate system view cycle architectural design fine grained system include logic thread switching result cost switch thread small important note resource physical core cache pipeline share hardware thread processing core execute hardware thread time consequently multithreaded multicore processor actually require different level scheduling show figure 5.15 illustrate dual threaded process- level scheduling decision oper- ating system choose software thread run hardware thread logical cpu practical purpose decision primary focus chapter level scheduling oper- ating system choose scheduling algorithm include describe section 5.3 second level scheduling specify core decide hard- ware thread run strategy adopt situation approach use simple round robin algorithm schedule hardware thread process core approach adopt ultrasparc t3 approach intel itanium dual core processor hardware manage thread core approach intel itanium dual core processor hardware manage thread core assign hardware thread dynamic urgency value range 0 7 0 represent low urgency 7 high itanium identify different event level scheduling trigger thread switch event occur thread switch logic compare urgency thread select thread high urgency value execute processor core note different level scheduling show figure 5.15 necessarily mutually exclusive fact operate system scheduler level aware sharing processor resource effective scheduling decision example assume cpu processing core core hardware thread software thread run system run core separate core schedule run core share processor resource likely proceed slowly schedule separate core operate system aware level processor resource sharing schedule software thread logical processor share resource smp system important workload balanced processor fully utilize benefit have processor oth- erwise processor sit idle processor high workload ready queue thread await cpu load balanc- e attempt workload evenly distribute processor smp system important note load balancing typically necessary system processor private ready queue eligi- ble thread execute system common run queue load balancing unnecessary processor idle immediately extract runnable thread common ready queue general approach load balancing push migration pull migration push migration specific task periodically check load processor find imbalance evenly distribute load move push thread overloaded idle busy processor pull migration occur idle processor pull wait task busy processor push pull migration need mutually exclusive fact implement parallel load balance system example linux cfs scheduler describe section 5.7.1 ule scheduler available freebsd system implement technique concept balanced load different meaning view balanced load require simply queue approximately number thread alternatively balance require equal distri- bution thread priority queue addition certain situation strategy sufficient work goal scheduling algorithm leave consideration consider happen cache memory thread run specific processor datum recently access thread populate cache processor result successive memory access thread satisfy cache memory cache processor result successive memory access thread satisfy cache memory know warm cache consider happen thread migrate processor load balancing content cache memory invalidate pro- cessor cache second processor repopulate high cost invalidate repopulate cache operate system smp support try avoid migrate thread processor instead attempt thread run processor advantage warm cache know processor affinit process affinity processor currently run strategy describe section 5.5.1 organize queue thread available scheduling implication processor affinity adopt approach common ready queue thread select execution processor thread schedule new processor processor cache repopulate private processor ready queue thread schedule processor benefit content warm cache essentially processor ready queue provide processor affinity free processor affinity take form operate system policy attempt process run processor guarantee situation know soft affinit operate system attempt process single processor possible process migrate processor load balancing contrast system provide system call support hard affinit allow process specify subset processor run system provide soft hard affinity example linux implement soft affinity provide sched setaffinity system support hard affinity allow thread specify set cpu eligible run main memory architecture system affect processor affinity issue figure 5.16 illustrate architecture feature non uniform memory access numa physical processor chip cpu local memory system interconnect allow cpu numa system share physical address space cpu fast access local memory memory local cpu operate system cpu scheduler memory placement algorithm numa aware numa cpu scheduling work thread schedule particular cpu allocate memory close cpu reside provide thread fast possible memory access interestingly load balancing counteract benefit processor affinity benefit keep thread run processor thread advantage datum processor cache memory balance load move thread processor remove benefit similarly migrate thread processor incur penalty numasystems thread move processor require long memory access time word natural tension load balancing minimize memory access time scheduling algorithm modern multicore numasystems complex section 5.7.1

access time scheduling algorithm modern multicore numasystems complex section 5.7.1 examine linux cfs scheduling algorithm explore balance compete goal example discuss far processor identical term capability allow thread run processing core difference memory access time vary base load balancing processor affinity policy numa system mobile system include multicore architecture sys- tem design core run instruction set vary term clock speed power management include ability adjust power consumption core point idle core system know heterogeneous multiprocessing hmp note form asymmetric multiprocessing describe section 5.5.1 system user task run core intention hmp well manage power consumption assign task certain core base specific demand task arm processor support type architecture know big little high peformance big core combine energy efficient little core big core consume great energy real time cpu scheduling short period time likewise little core use energy long period advantage approach combine number slow core fast one cpu scheduler assign task require high performance need run long period background task little core help preserve battery charge similarly interactive application require processing power run short duration assign big core additionally mobile device power save mode energy intensive big core disable system rely solely energy efficient little core win- dows 10 support hmp scheduling allow thread select scheduling policy well support power management demand real time cpu scheduling cpu scheduling real time operate system involve special issue general distinguish soft real time system hard real time system soft real time system provide guarantee critical real time process schedule guarantee process give preference noncritical process hard real time system strict requirement task service deadline service deadline expire service section explore issue relate process scheduling soft hard real time consider event drive nature real time system system typically wait event real time occur event arise software timer expire hardware remote control vehicle detect approach obstruction

software timer expire hardware remote control vehicle detect approach obstruction event

occur system respond service quickly possible refer event latency time elapse event occur service figure 5.17 event e occur real time system respond e usually different event different latency requirement example latency requirement antilock brake system 3 5 millisec- ond time wheel detect slide system control antilock brake 3 5 millisecond respond control situation response take long result automobile veer control contrast embed system control radar airliner tolerate latency period second type latency affect performance real time system 1 interrupt latency 2 dispatch latency interrupt latency refer period time arrival interrupt cpu start routine service interrupt interrupt occur operate system complete instruction execute determine type interrupt occur save state current process service interrupt specific interrupt service routine isr total time require perform task interrupt latency figure 5.18 obviously crucial real time operate system minimize inter- rupt latency ensure real time task receive immediate attention hard real time system interrupt latency simply minimize bound meet strict requirement system important factor contribute interrupt latency time interrupt disable kernel datum structure update real time operate system require interrupt disable short period time time require scheduling dispatcher stop process start know dispatch latency provide real time task t run real time cpu scheduling response event task immediate access cpu mandate real time operate system minimize latency effective technique keep dispatch latency low provide preemptive kernel hard real time system dispatch latency typically measure microsecond figure 5.19 diagram makeup dispatch latency conflic phase dispatch latency component 1 preemption process run kernel 2 release low priority process resource need high priority follow conflict phase dispatch phase schedule high priority process available cpu important feature real time operate system respond imme- diately real time process soon process require cpu result scheduler real time operate system support

process require cpu result scheduler real time operate system support priority- base algorithm preemption recall priority base scheduling algo- rithm assign process priority base importance important task assign high priority deem important scheduler support preemption process currently run cpu preempt high priority process available run preemptive priority base scheduling algorithm discuss detail section 5.3.4 section 5.7 present example soft real time schedul- e feature linux windows solaris operating system system assign real time process high scheduling priority example windows 32 different priority level high level priority value 16 31 reserve real time process solaris linux similar prioritization scheme note provide preemptive priority base scheduler guaran- tee soft real time functionality hard real time system guarantee real time task service accord deadline requirement make guarantee require additional scheduling feature remainder section cover scheduling algorithm appropriate hard proceed detail individual scheduler define certain characteristic process schedule process consider periodic require cpu constant interval period periodic process acquire cpu fix processing time t deadline d service cpu period p. relationship processing time deadline period express 0 t d p rate periodic task 1p figure 5.20 illustrate execution periodic process time scheduler advantage characteristic assign priority accord process deadline rate requirement unusual form scheduling process announce deadline requirement scheduler technique know admission control algorithm scheduler thing admit process guarantee process complete time reject request impossible guarantee task service deadline rate monotonic scheduling algorithm schedule periodic task static priority policy preemption low priority process run- ne high priority process available run preempt low priority process enter system periodic task assign priority inversely base period short period high priority long period low priority rationale policy assign high priority task require cpu furthermore rate monotonic scheduling assume process- real time cpu scheduling scheduling task p2 high priority p1 e time periodic process cpu burst time process acquire

p1 e time periodic process cpu burst time process acquire cpu duration cpu burst let consider example process p1 p2 period p1 p2 50 100 respectively p1 = 50 p2 = 100 processing time t1 = 20 p1 t2 = 35 p2 deadline process require complete cpu burst start period ask possible schedule task meet deadline measure cpu utilization process pi ratio burst period tipi cpu utilization p1 2050 = 0.40 p2 35100 = 0.35 total cpu utilization 75 percent schedule task way meet deadline leave cpu available cycle suppose assign p2 high priority p1 execution p1 p2 situation show figure 5.21 p2 start execution complete time 35 point p1 start complete cpu burst time 55 deadline p1 time 50 scheduler cause p1 miss deadline suppose use rate monotonic scheduling assign p1 high priority p2 period p1 short p2 execution process situation show figure 5.22 p1 start complete cpu burst time 20 meet deadline p2 start run point run time 50 time preempt p1 5 millisecond remain cpu burst p1 complete cpu burst time 70 point scheduler resume p2 p2 complete cpu burst time 75 meet deadline system idle time 100 p1 schedule rate monotonic scheduling consider optimal set pro- cesse schedule algorithm schedule algorithm assign static priority let examine set pro- cesse schedule rate monotonic algorithm assume process p1 period p1 = 50 cpu burst t1 = 25 p2 correspond value p2 = 80 t2 = 35 rate monotonic 120 130 140 150 160 170 180 190 200 90 100 110

miss deadline rate monotonic scheduling scheduling assign process p1 high priority short period total cpu utilization process 2550 + 3580 = 0.94 logical process schedule leave cpu 6 percent available time figure 5.23 show scheduling process p1 p2 initially p1 run complete cpu burst time 25 process p2 begin run run time 50 preempt p1 point p2 10 millisecond remain cpu burst process p1 run time 75 consequently p2 finish burst time 85 deadline completion cpu burst time 80 despite optimal rate monotonic scheduling limitation cpu utilization bound possible maximize cpu resource fully bad case cpu utilization schedule n process process system cpu utilization 100 percent fall approximately 69 percent number process approach infinity process cpu utilization bound 83 percent combined cpu utilization process schedule figure 5.21 figure 5.22 75 percent rate monotonic scheduling algorithm guarantee schedule meet deadline process schedule figure 5.23 combined cpu utilization approximately 94 per- cent rate monotonic scheduling guarantee schedule meet deadline early deadline fir edf scheduling assign priority dynamically accord- e deadline early deadline high priority late deadline low priority edf policy process runnable announce deadline requirement system priority adjust reflect deadline newly runnable process note differ rate monotonic scheduling priority illustrate edf scheduling schedule process show figure 5.23 fail meet deadline requirement rate monotonic scheduling recall p1 value p1 = 50 t1 = 25 p2 value p2 = 80 t2 = 35 edf scheduling process show figure 5.24 process p1 early deadline initial priority high process p2 process p2 begin run end cpu burst p1 rate monotonic scheduling allow p1 preempt p2 real time cpu scheduling beginning period time 50 edf scheduling allow process p2 continue run p2 high priority p1 deadline time 80 early p1 time 100 p1 p2

meet deadline process p1 begin run time 60 complete second cpu burst time 85 meet second deadline time 100 p2 begin run point preempt p1 start period time 100 p2 preempt p1 early deadline time 150 p2 time 160 time 125 p1 complete cpu burst p2 resume execution finish time 145 meet deadline system idle time 150 p1 schedule run unlike rate monotonic algorithm edf scheduling require process periodic process require constant cpu time burst requirement process announce deadline scheduler runnable appeal edf scheduling theoretically optimal theoretically schedule process process meet deadline requirement cpu utilization 100 percent practice impossible achieve level cpu utilization cost context switching process interrupt proportional share scheduling proportional share scheduler operate allocate t share application application receive n share time ensure application nt total processor time example assume total t = 100 share divide process b c. assign 50 share b assign 15 share c assign 20 share scheme ensure 50 percent total processor time b 15 percent c 20 percent proportional share scheduler work conjunction admission control policy guarantee application receive allocate share time admission control policy admit client request particular number share sufficient share available current example allocate 50 + 15 + 20 = 85 share total 100 share new process d request 30 share admission controller deny d entry system posix real time scheduling posix standard provide extension real time computing posix.1b cover posix api relate schedule real time thread posix define scheduling class real time thread sched fifo sched rr sched fifo schedule thread accord come serve policy fifo queue outline section 5.3.1

time slic- e thread equal priority high priority real time thread fifo queue grant cpu termi- nate block sched rr use round robin policy similar sched fifo provide time slicing thread equal priority posix provide additional scheduling class

time slicing thread equal priority posix provide additional scheduling class sched implemen- tation undefined system specific behave differently different posix api specify following function get set scheduling policy pthread attr getschedpolicy(pthread attr t attr int pthread attr setschedpolicy(pthread attr t attr int parameter function pointer set attribute thread second parameter 1 pointer integer set current scheduling policy

pthread attr getsched policy 2 integer value sched fifo sched rr sched pthread attr setsched policy function function return nonzero value error occur figure 5.25 illustrate posix pthread program api program determine current scheduling policy set scheduling algorithm sched fifo turn description scheduling policy linux win- dow solaris operating system important note use term process scheduling general sense fact describe scheduling kernel thread solaris windows system task linux scheduler example linux scheduling process scheduling linux interesting history prior version 2.5 linux kernel run variation traditional unix scheduling algorithm algorithm design smp system mind adequately support system multiple processor addition result poor performance system large number runnable pro- cesse version 2.5 kernel scheduler overhaul include scheduling algorithm know o(1)—that run constant time regard- number task system o(1 scheduler provide define num thread 5 int main(int argc char argv int policy pthread t tid[num thread pthread attr t attr default attribute pthread attr init(&attr current scheduling policy pthread attr getschedpolicy(&attr policy = 0 fprintf(stderr unable policy.n policy = = sche policy = = sche rr policy = = sche fifo set scheduling policy fifo rr pthread attr setschedpolicy(&attr sched fifo = 0 fprintf(stderr unable set policy.n create thread = 0 < num thread i++ join thread = 0 < num thread i++ pthread join(tid[i null thread begin control function void runner(void param work posix real time scheduling api increase support smp system include processor affinity

load bal- ancing processor practice o(1 scheduler deliver excellent performance smp system lead poor response time interactive process common desktop computer sys- tem development 2.6 kernel scheduler revise release 2.6.23 kernel completely fair scheduler cfs default linux scheduling algorithm scheduling linux system base scheduling class class assign specific priority different scheduling class kernel accommodate different scheduling algorithm base need system process scheduling criterion linux server exam- ple different mobile device run linux decide task run scheduler select high priority task belong- e high priority scheduling class standard linux kernel implement scheduling class 1 default scheduling class cfs scheduling algorithm 2 real time scheduling class discuss class new scheduling class course add strict rule associate relative priority value length time quantum cfs scheduler assign proportion cpu processing time task proportion calculate base nice value assign task nice value range 20 +19 numerically low nice value indicate high relative priority task low nice value receive high proportion cpu processing time task high nice value default nice value 0 term nice come idea task increase nice value 0 +10 nice task system lower relative priority word nice process finish cfs use discrete value time slice instead identify target latency interval time runnable task run proportion cpu time allocate value target latency addition have default minimum value target latency increase number active task system grow certain threshold cfs scheduler directly assign priority record long task run maintain virtual run time task task variable vruntime virtual run time associate decay factor base priority task low priority task high rate decay high priority task task normal priority nice value 0

virtual run time identical actual physical run time task default priority run 200 millisecond vruntime 200 millisecond low priority task run 200 millisecond vruntime high 200 millisecond similarly high priority task run 200 millisecond vruntime 200 millisecond decide task run scheduler simply select task small vruntime value addition high priority task available run preempt low priority task let examine cfs scheduler action assume task nice value task o bound cpu bound typically o bind task run short period block additional o cpu bind task exhaust time period opportunity run processor value vruntime linux cfs scheduler provide efficient algorithm select task run standard queue datum structure runnable task place red black tree balanced binary search tree key base value vruntime tree show task small value vruntime value vruntime task runnable add tree task tree runnable example block wait o remove generally speak task give processing time small value vruntime left tree task give processing time right accord property binary search tree leftmost node small key value sake cfs scheduler mean task high priority red black tree balance navigate discover leftmost node require o(log n operation n number node tree efficiency reason linux scheduler cache value variable rb leftmost determine task run require retrieve cache value eventually low o bind task cpu bind task give o bind task high priority cpu bind task point cpu bind task execute o bind task eligible run example o task wait available o bind task preempt cpu bind task linux implement real time scheduling posix standard describe section 5.6.6 task schedule sched fifo sched rr real time policy run high priority normal non real- time task linux use separate priority range real time task second normal task real time task assign static priority range 0 99

normal task assign priority 100 139 range map global priority scheme numerically low value indicate high relative priority normal task assign priority scheduling priority linux system base nice value value 20 map priority 100 nice value +19 map 139 scheme show figure 5.26 cfs scheduler support load balancing sophisticated technique equalize load processing core numa- aware minimize migration thread cfs define load thread combination thread priority average rate cpu utilization thread high priority o bind require little cpu usage generally low load similar load low priority thread high cpu utilization metric load queue sum load thread queue balancing simply ensure queue approximately load highlight section 5.5.4 migrate thread result memory access penalty have invalidate cache con- tents numa system incur long memory access time address problem linux identify hierarchical system scheduling domain scheduling domain set cpu core balance idea illustrate figure 5.27 core scheduling domain group accord share resource system example core show figure 5.27 level 1 l1 cache pair core share level 2 l2 cache organize separate domain0 domain1 likewise domain share level 3 l3 cache organize processor level domain know numa node take step numasystem physical processor domain numa aware load balancing linux cfs scheduler large system level domain combine separate processor level numa general strategy cfs balance load domain begin- ne low level hierarchy figure 5.27 example initially thread migrate core domain i.e. domain0 domain1 load balancing level occur domain0 domain1 cfs reluctant migrate thread sep- arate numa node thread move far local memory migration occur severe load imbalance general rule overall system busy cfs load balance domain local core avoid memory latency penalty numa example windows schedule windows schedule thread priority base preemptive scheduling algorithm windows scheduler ensure high priority thread run portion windows kernel handle scheduling call dispatcher thread select run dispatcher run preempt high

scheduling call dispatcher thread select run dispatcher run preempt high priority thread terminate time quantum end call block system o. high priority real time thread ready low priority thread run low priority thread preempt preemption give real time thread preferential access cpu thread need dispatcher use 32 level priority scheme determine order thread execution priority divide class variable class contain thread have priority 1 15 real time class contain thread priority range 16 31 thread run priority 0

memory management dispatcher use queue scheduling priority traverse set queue high low find thread ready run ready thread find dispatcher execute special thread call idle thread relationship numeric priority windows kernel windows api windows api identify following priority class process belong idle priority class normal priority class normal priority class normal priority class high priority class realtime priority class process typically member normal priority class process belong class parent process member idle priority class class specify process create additionally priority class process alter setpriorityclass function windows api priority class realtime priority class variable mean priority thread belong class change athread give priority class relative priority value relative priority include normal normal time critical priority thread base priority class belong relative priority class relationship show figure 5.28 value priority class appear row left column contain value relative priority example relative priority thread normal priority class normal numeric priority thread 10 furthermore thread base priority represent value priority range class thread belong default base priority value normal relative priority class base priority priority class follow realtime priority class—24 high priority class—13 normal priority class—10 normal priority class—8 windows thread priority normal priority class—6 idle priority class—4 initial priority thread typically base priority process thread belong setthreadpriority function windows api modify thread base priority thread time quantum run thread interrupt thread variable priority class priority lower priority lower base priority lower priority tend limit cpu consumption compute bind thread variable- priority thread release wait operation dispatcher boost priority boost depend thread wait example thread wait keyboard o large increase thread wait disk operation moderate strategy tend good response time interactive thread mouse window enable o bind thread o device busy permit compute bind thread use spare cpu cycle background strategy operate system include unix addition window user currently interact receive priority boost enhance

unix addition window user currently interact receive priority boost enhance response time user run interactive program system need pro- vide especially good performance reason windows special scheduling rule process normal priority class windows distin- guishe

foreground process currently select screen background process currently select process move foreground windows increase scheduling quantum factor typically 3 increase give foreground process time long run time sharing preemption occur windows 7 introduce user mode scheduling ums allow appli- cation create manage thread independently kernel application create schedule multiple thread involve windows kernel scheduler application create large number thread schedule thread user mode efficient kernel- mode thread scheduling kernel intervention necessary early version windows provide similar feature know fiber allow user mode thread fiber map single kernel thread fiber limited practical use afiber unable call windows api fiber share thread environment block teb thread run pre- sente problem windows api function place state information teb fiber information overwrite different fiber ums overcome obstacle provide user mode thread addition unlike fiber ums intend directly programmer detail write user mode scheduler chal- lenge ums include scheduler scheduler come programming language library build ums example microsoft provide concurrency runtime concrt concurrent programming framework c++ design task base parallelism section 4.2 multicore processor concrt provide user mode scheduler facility decompose program task schedule available processing core support schedule multiprocessor system describe section 5.5 attempt schedule thread optimal processing core thread include maintain thread preferred recent processor technique windows create set logical processor know smt set hyper threaded smt system hardware thread belong cpu core belong smt set logical processor number begin 0 example dual threaded quad core system contain logical processor consist smt set 0 1 2 3 4 5 6 7

avoid cache memory access penalite highlight section 5.5.4 scheduler attempt maintain thread run logical processor smt set distribute load different logical processor thread assign ideal processor number represent thread prefer processor process initial seed value identify ideal cpu thread belong process seed incremente new thread create process distribute load different logical processor smt system increment ideal processor smt set example dual threaded quad core system ideal processor thread specific process assign 0 2 4 6 0 2 avoid situation wherby thread process assign processor 0 process assign different seed value distribute load thread physical processing core system continue example seed second process

1 ideal processor assign order 1 3 5 7 1 3 example solaris scheduling solaris use priority base thread scheduling thread belong 1 time sharing ts 2 interactive ia 3 real time rt 4 system sys 5 fair share fss 6 fixed priority fp class different priority different scheduling algo- default scheduling class process time sharing scheduling policy time sharing class dynamically alter priority assign time slice different length multilevel feedback queue default inverse relationship priority time slice high solaris dispatch table time sharing interactive thread priority small time slice low priority large time slice interactive process typically high priority cpu bind process low priority scheduling policy give good response time interactive process good throughput cpu bind process interactive class use scheduling policy time sharing class give windowing application create kde gnome window manager high priority well performance figure 5.29

show simplified dispatch table schedule time sharing interactive thread scheduling class include 60 priority level brevity display handful dispatch table solaris system vm run dispadmin -c ts -g dispatch table show figure 5.29 contain following field priority class dependent priority time sharing interactive class high number indicate high priority time quantum time quantum associate priority illus- trate inverse relationship priority time quanta low priority priority 0

high time quantum 200 millisec- ond high priority priority 59 low time quantum time quantum expire new priority thread entire time quantum block thread consider cpu- intensive show table thread priority low- return sleep priority thread return sleep wait o table illustrate o available wait thread priority boost 50 59 support- ing scheduling policy provide good response time interactive thread real time class give high priority real time process run process class assignment allow real time process guaranteed response system bound period time general process belong solaris use system class run kernel thread scheduler paging daemon priority system thread establish change system class reserve kernel use user process run kernel mode system class fix priority fair share class introduce solaris 9 thread fix priority class priority range time sharing class priority dynamically adjust fair share class use cpu share instead priority scheduling decision cpu share indicate entitlement available cpu resource allocate set process know project scheduling class include set priority scheduler convert class specific priority global priority select thread high global priority run select thread run cpu 1 block 2 use time slice 3 preempt high priority thread multiple thread priority scheduler use round robin queue figure 5.30 illustrate scheduling class relate map global priority notice kernel maintain thread service interrupt thread belong scheduling class execute high priority 160–169 mention solaris traditionally model section 4.3.3 switch model section 4.3.2 begin select cpu scheduling algorithm particular system see section 5.3 scheduling algorithm parameter result select algorithm difficult problem define criterion select algo- rithm see section 5.2 criterion define term cpu utilization response time throughput select algorithm define relative importance element criterion include measure maximize cpu utilization constraint maximum response time 300 millisecond realtime rt thread system sys thread fair share fss thread fixed priority fx thread timeshare ts thread interactive ia

thread fixed priority fx thread timeshare ts thread interactive ia thread maximize throughput turnaround time average linearly proportional total execution time selection criterion define want evaluate algo- rithm consideration describe evaluation method use major class evaluation method analytic evaluation analytic evalu- ation use give algorithm system workload produce formula number evaluate performance algorithm workload deterministic modeling type analytic evaluation method take particular predetermined workload define performance algorithm workload example assume workload show process arrive time 0 order give length cpu burst give millisecond consider fcfs sjf rr quantum = 10 millisecond scheduling algo- rithm set process algorithm minimum average waiting time fcfs algorithm execute process waiting time 0 millisecond process p1 10 millisecond process p2 39 millisecond process p3 42 millisecond process p4 49 millisecond process p5 average waiting time 0 + 10 +

39 + 42 + 49)/5 = 28 millisecond nonpreemptive sjf scheduling execute process waiting time 10 millisecond process p1 32 millisecond process p2 0 millisecond process p3 3 millisecond process p4 20 millisec- ond process p5 average waiting time 10 + 32 + 0 + 3 + 20)/5 = 13 millisecond rr algorithm execute process waiting time 0 millisecond process p1 32 millisecond process p2 20 millisecond process p3 23 millisecond process p4 40 millisecond process p5 average waiting time 0 + 32 + 20 + 23 + 40)/5 = 23 millisecond case average waiting time obtain sjf policy half obtain fcfs scheduling rr algorithm give intermediate value deterministic modeling simple fast give exact number allow- e compare algorithm require exact number input answer apply case main use deterministic modeling describe scheduling algorithm provide example case run program measure program processing requirement exactly able use deterministic modeling select scheduling algorithm furthermore set example deterministic modeling indicate trend analyze prove separately example show environment describe process time

trend analyze prove separately example show environment describe process time available time 0 sjf policy result minimum waiting time system process run vary day day static set process time use deterministic modeling determine distribution cpu o burst dis- tribution measure approximate simply estimate result mathematical formula describe probability particular cpu burst commonly distribution exponential describe mean similarly describe distribution time process arrive system arrival time distribution distribution possible compute average throughput utilization waiting time algorithm computer system describe network server server queue wait process cpu server ready queue o system device queue know arrival rate service rate compute utilization average queue length average wait time area study call queueing network analysis example let n average long term queue length exclude process service let w average waiting time queue let λ average arrival rate new process queue process second expect time w process wait λ × w new process arrive queue system steady state number process leave queue equal number process arrive n = λ × w. equation know little formula particularly useful valid scheduling algorithm arrival distribution example n number customer store use little formula compute variable know example know 7 process arrive second average normally 14 process queue compute average waiting time process 2 second queueing analysis useful compare scheduling algorithm limitation moment class algorithm distribu- tion handle fairly limited mathematic complicated algorithm distribution difficult work arrival service distribution define mathematically tractable unre- alistic way generally necessary number indepen- dent assumption accurate result difficulty queueing model approximation real system accu- racy compute result questionable accurate evaluation scheduling algorithm use simu- lation run simulation involve program model computer system software datum structure represent major component sys- tem simulator variable represent clock variable value increase simulator modify system state reflect activity device process scheduler simulation execute statistic indicate algorithm performance gather

device process scheduler simulation execute statistic indicate algorithm performance gather print datum drive simulation generate way common method use random number generator program generate process cpu burst time arrival departure accord probability distribution distribution define mathematically uniform exponential poisson empirically distribution define empirically measurement actual system study take result define distribution event real system distribution drive simulation distribution drive simulation inaccurate relationship successive event real system frequency distribution indicate instance event occur indicate order occurrence correct problem use trace file create trace monitor real system record sequence actual event figure 5.31 use sequence drive simulation trace file provide excellent way compare algorithm exactly set real input method produce accurate result input simulation expensive require hour computer time detailed simulation provide accurate result rr q = 14 rr q = 14 evaluation cpu scheduler simulation take computer time addition trace file require large amount storage space finally design coding debugging simulator major task simulation limited accuracy completely accurate way evaluate scheduling algorithm code operate system work approach put actual algorithm real system evaluation real operating condition method expense expense incur code algorithm modify operate system support required datum structure cost test change usually virtual machine dedicated hardware regression testing confirm change bad cause new bug cause old bug recreate example algorithm replace solve bug change cause bug reoccur difficulty environment algorithm change environment change usual way new program write type problem change result performance scheduler short process give priority user break large process set small process interactive process give priority noninteractive process user switch interactive use problem usually address tool script encapsulate complete set action repeatedly tool tool measure result detect problem cause new environment course human program behavior attempt circumvent schedul- e algorithm example researcher design system classi- fie interactive noninteractive process automatically look terminal o. process input output

interactive noninteractive process automatically look terminal o. process input output terminal 1 second interval process classify noninteractive move low priority queue response policy programmer modify program write arbitrary character terminal regular interval 1 second system give program high priority terminal output completely meaningless general flexible scheduling algorithm alter system manager user tune spe- cific application set application workstation perform high end graphical application instance scheduling need different web server file server operate system particularly sev- eral version unix allow system manager fine tune scheduling parameter particular system configuration example solaris pro- vide dispadmin command allow system administrator modify parameter scheduling class describe section 5.7.3 approach use api modify priority process thread java posix windows api provide function downfall approach performance tune system application result improve performance general situation cpu scheduling task select wait process ready queue allocate cpu cpu allocate select process dispatcher scheduling algorithm preemptive cpu take away process nonpreemptive process voluntarily relinquish control cpu modern operate system preemptive scheduling algorithm evaluate accord following criterion 1 cpu utilization 2 throughput 3 turnaround time 4 waiting time 5 response time come serve fcfs scheduling simple scheduling algo- rithm cause short process wait long process short job sjf scheduling provably optimal provide short- est average waiting time implement sjf scheduling difficult how- predict length cpu burst difficult round robin rr scheduling allocate cpu process time quantum process relinquish cpu time quan- tum expire process preempt process schedule run time quantum priority scheduling assign process priority cpu allocate process high priority process priority schedule fcfs order rr scheduling multilevel queue scheduling partition process separate queue arrange priority scheduler execute process high priority queue different scheduling algorithm queue multilevel feedback queue similar multilevel queue process migrate different queue multicore processor place cpu physical chip cpu hardware thread per- spective operate system hardware thread

chip cpu hardware thread per- spective operate system hardware thread appear load balancing multicore system equalize load cpu core migrate thread core balance load invalidate cache content increase memory access time soft real time scheduling give priority real time task non real- time task hard real time scheduling provide timing guarantee real- rate monotonic real time scheduling schedule periodic task static priority policy preemption early deadline edf scheduling assign priority accord deadline early deadline high priority late deadline low priority proportional share scheduling allocate t share application application allocate n share time ensure have nt total processor time linux use completely fair scheduler

cfs assign proportion cpu processing time task proportion base virtual runtime vruntime value associate task windows scheduling use preemptive 32 level priority scheme deter- order thread scheduling solaris identify unique scheduling class map global priority cpu intensive thread generally assign low priority long time quantum o bind thread usually assign high priority short time quantum modeling simulation evaluate cpu scheduling algo- cpu scheduling algorithm determine order execution schedule process give n process schedule proces- sor different schedule possible formula term explain difference preemptive nonpreemptive schedul- suppose following process arrive execution time indicate process run time list answer- e question use nonpreemptive scheduling base decision information time decision average turnaround time process fcfs scheduling algorithm average turnaround time process sjf scheduling algorithm sjf algorithm suppose improve performance notice choose run process p1 time 0 know short process arrive soon compute average turnaround time cpu leave idle 1 unit sjf scheduling remember process p1 p2 wait idle time waiting time increase algorithm know future knowledge consider follow set process length cpu burst time give millisecond process assume arrive order p1 p2 p3 p4 p5 time 0

draw gantt chart illustrate execution pro- cesse following scheduling algorithm fcfs sjf non- preemptive priority large priority number imply high priority rr quantum = 2 turnaround time process scheduling algorithm waiting time process schedul- algorithm result minimum average waiting time process following process schedule preemptive round- robin scheduling algorithm process assign numerical priority high number indi- cat high relative priority addition process list system idle task consume cpu resource identify pidle task priority 0 schedule when- system available process run length time quantum 10 unit process preempt high priority process preempt process place end queue scheduling order process gantt chart turnaround time process waiting time process cpu utilization rate advantage have different time quantum size dif- ferent level multilevel queueing system cpu scheduling algorithm parameterize example rr algorithm require parameter indicate time slice multilevel feedback queue require parameter define number queue scheduling algorithm queue criterion process queue algorithm set algorithm example set rr algorithm time slice set algorithm include example fcfs algorithm rr algorithm infinite time quantum relation hold following pair algorithm set priority sjf multilevel feedback queue fcfs priority fcfs rr sjf suppose cpu scheduling algorithm favor process processor time recent past algorithm favor o bind program permanently starve distinguish pcs scs scheduling traditional unix scheduler enforce inverse relationship priority number priority high number low priority scheduler recalculate process priority second following function priority = recent cpu usage 2 + base base = 60 recent cpu usage refer value indicate process cpu priority recalculate assume recent cpu usage

process p1 40 process p2 18 process p3 10 new priority process priority recalculate base information traditional unix scheduler raise lower relative priority cpu bind process scheduling policy freebsd 5.2 present mckusick et al 2015 linux cfs scheduler describe solaris scheduling describe mauro mcdougall 2007 russi- novich et al 2017 discuss scheduling windows internal butenhof 1997 lewis berg 1998 describe scheduling pthreads system multicore scheduling examine mcnairy bhatia 2005 kongetira et al 2005 siddha et al 2007 d. butenhof program posix threads addison- kongetira et al 2005 p. kongetira k. aingaran k. olukotun niagara 32 way multithreaded sparc processor ieee micro magazine volume 25 number 2 2005 page 21–29 lewis berg 1998 b. lewis d. berg multithreaded programming pthreads sun microsystems press 1998 mauro mcdougall 2007 j. mauro r. mcdougall solaris internals core kernel architecture prentice hall 2007 mckusick et al 2015 m. k. mckusick g. v. neville neil r. n. m. wat- son design implementation freebsd unix operating system second edition pearson 2015 mcnairy bhatia 2005 c. mcnairy r. bhatia montecito dual core dual thread itanium processor ieee micro magazine volume 25 number 2 2005 page 10–20 russinovich et al 2017 m. russinovich d. a. solomon a. ionescu win- dows

internals 1 seventh edition microsoft press 2017 siddha et al 2007 s. siddha v. pallipadi a. mallick process schedul- ing challenge era multi core processor intel technology journal volume 11 number 4 2007 chapter 5 exercise type program likely voluntary context switch likely nonvoluntary context switch explain discuss following pair scheduling criterion conflict certain cpu utilization response time average turnaround time maximum waiting time o device utilization cpu utilization technique implement lottery scheduling work assign process lottery ticket allocate cpu time when- scheduling decision lottery ticket choose random process hold ticket get cpu btv oper- ating system implement lottery scheduling hold lottery 50 time second lottery winner get 20 millisecond cpu time 20 millisecond × 50 = 1 second describe btv scheduler ensure high priority thread receive attention cpu low priority thread scheduling algorithm maintain run queue list process eligible run processor multicore system general option 1 processing core run queue 2 single run queue share processing core advantage disadvantage approach consider exponential average formula predict length cpu burst implication assign following value parameter algorithm $\alpha = 0$ $\tau 0 = 100$ millisecond $\alpha = 0.99$ $\tau 0 = 10$ millisecond variation round robin scheduler regressive round robin scheduler scheduler assign process time quantum priority initial value time quantum 50 millisecond time process allocate cpu use entire time quantum block o 10 millisecond add time quantum priority level boost time quantum process increase maximum 100 millisecond process block entire time quantum time quantum reduce 5 millisecond priority remain type process cpu bind o bound regressive round robin scheduler favor explain consider following set process length cpu burst give millisecond process assume arrive order p1 p2 p3 p4 p5 time 0 draw gantt chart illustrate execution pro- cesse following scheduling algorithm fcfs sjf non- preemptive priority large priority number imply high priority rr

non- preemptive priority large priority number imply high priority rr quantum = 2 turnaround time process scheduling algorithm waiting time process schedul- algorithm result minimum average waiting time process follow process schedule preemptive priority base round robin scheduling algorithm process assign numerical priority high number indi- cat high relative priority scheduler execute highest- priority process process priority round robin scheduler time quantum 10 unit process preempt high priority process preempt process place end queue scheduling order process gantt chart turnaround time process waiting time process nice command set nice value process linux unix system explain system allow user assign process nice value > = 0 allow root administrator user assign nice value < 0 following scheduling algorithm result starvation short job consider variant rr scheduling algorithm entry ready queue pointer pcb effect put pointer process ready queue major advantage disadvantage modify basic rr algorithm achieve effect duplicate pointer consider system run o bind task cpu bind task assume o bind task issue o operation millisecond cpu computing o operation take 10 millisecond complete assume context switch overhead 0.1 millisecond process long run task describe cpu utilization round robin scheduler time quantum 1 millisecond time quantum 10 millisecond consider system implement multilevel queue scheduling strategy computer user employ maximize cpu time allocate user process consider preemptive priority scheduling algorithm base dynami- cally change priority large priority number imply high priority process wait cpu ready queue run- ning priority change rate α run priority change rate β process give priority 0 enter ready queue parameter α β set different scheduling algorithm algorithm result $\beta > \alpha > 0$ algorithm result $\alpha < \beta < 0$

explain following scheduling algorithm discriminate favor short process multilevel feedback queue describe share ready queue suffer performance problem smp environment consider load balance algorithm ensure queue approximately number thread independent priority effectively priority base scheduling algorithm handle situation run queue high priority thread second queue low priority thread assume smp system private processor run queue new process create place queue parent process separate queue benefit place new process queue parent benefit place new process different assume thread block network o eligible run describe numa aware scheduling algorithm reschedule thread cpu previously run windows scheduling algorithm determine numeric pri- ority follow thread thread realtime priority class relative priority thread normal priority class relative priority high thread normal priority class relative priority normal assume thread belong realtime priority class assign time critical priority combination priority class priority correspond high possible relative priority windows scheduling consider scheduling algorithm solaris operating system time quantum millisecond thread pri- ority 15 priority 40 assume thread priority 50 entire time quantum block new priority scheduler assign thread assume thread priority 20 block o time quantum expire new priority scheduler assign assume task b run linux system nice value b 5 +5 respectively cfs scheduler guide describe respective value vruntime vary process give following scenario b cpu bind o bound b cpu bound cpu bind b o bound provide specific circumstance illustrate rate monotonic scheduling inferior early deadline scheduling meet real time process deadline consider process p1 p2 p1 = 50 t1 = 25 p2 = 75 t2 = 30 process schedule rate monotonic scheduling illustrate answer gantt chart one figure 5.21 figure 5.24

illustrate scheduling process earliest- deadline edf scheduling explain interrupt dispatch latency time bound hard real time system describe advantage heterogeneous multiprocessing project involve implement different process scheduling algo- rithm scheduler assign predefined set task schedule task base select scheduling algorithm task assign priority cpu burst following scheduling algorithm come serve fcfs schedule task order request cpu short job sjf schedule task order length task cpu burst priority scheduling schedule task base priority round robin rr scheduling task run time quantum remainder cpu burst priority round robin schedule task order priority use round robin scheduling task equal priority priority range 1 10 high numeric value indicate high relative priority round robin scheduling length time quantum implementation project complete c java program file support language provide source code download text support file read schedule task insert task list invoke scheduler schedule task form task priority cpu burst following example format t1 4 20 t2 2 25 t3 3 25 t4 3 15 t5 10 10 task t1 priority 4 cpu burst 20 millisecond forth assume task arrive time scheduler algorithm support high priority process preempt process low priority addition task place queue list particular order different strategy organize list task present section 5.1.2 approach place task single unordered list strategy task selection depend scheduling algorithm example sjf scheduling search list find task short cpu burst alternatively list order accord- e scheduling criterion priority strategy involve have separate queue unique priority show figure 5.7 approach briefly discuss section 5.3.6 worth highlight- e term list queue somewhat interchangeably queue specific fifo functionality list strict insertion deletion requirement likely find functionality general list suitable complete project ii c implementation detail file driver.c read schedule task insert task link list invoke process scheduler call schedule function schedule function execute task accord specify scheduling algorithm task select

schedule function execute task accord specify scheduling algorithm task select execution cpu determine pick- nexttask function execute invoke run function define cpu.c file amakefile determine specific scheduling algo- rithm invoke driver example build fcfs scheduler enter execute scheduler schedule task schedule.txt refer readme file source code download detail proceed sure familiarize source code provide makefile iii java implementation detail file driver.java read schedule task insert task java arraylist invoke process scheduler call schedule method following interface identify generic scheduling algorithm different scheduling algorithm implement public interface algorithm implementation scheduling algorithm public void schedule select task schedule public task picknettask schedule method obtain task run cpu invok- e picknexttask method execute task call static run method cpu.java class program run follow java driver fcfs schedule.txt refer readme file source code download detail proceed sure familiarize java source file provide source code download iv challenge additional challenge

present project 1 task provide scheduler assign unique task tid scheduler run smp environment cpu separately run scheduler possible race condition variable assign task identifier fix race condition atomic integer linux macos system sync fetch add function atomically increment integer value example following code sample atomically increment value 1 int value = 0 sync fetch add(&value,1 refer java api detail use atomicinteger class java program 2 calculate average turnaround time waiting time response time scheduling algorithm system typically consist hundred thou- sand thread run concurrently parallel thread share user datum operate system continuously update datum structure support multiple thread race condition exist access share datum control possibly result corrupt data value process synchronization involve tool control access share datum avoid race condition tool carefully incorrect use result poor system performance include c h p t e r cooperate process affect affect process execute system cooperate process directly share logical address space code datum allow share datum share memory message passing concurrent access share datum result datum inconsistency chapter discuss mechanism ensure orderly execution cooperate process share logical address space data consistency maintain describe critical section problem illustrate race condition illustrate hardware solution critical section problem memory barrier compare swap operation atomic variable demonstrate mutex lock semaphore monitor condition vari- able solve critical section problem evaluate tool solve critical section problem low- moderate- high contention scenario see process execute concurrently parallel sec- tion 3.2.2

introduce role process scheduling describe cpu scheduler switch rapidly process provide concurrent exe- cution mean process partially complete execution process schedule fact process interrupt point instruction stream processing core assign execute instruction process additionally section 4.2 introduce parallel execution instruction stream represent different process execute simultaneously separate processing core chap- ter explain concurrent parallel execution contribute issue involve integrity datum share process let consider example happen chapter 3 devel- ope model system consist cooperate sequential process thread run asynchronously possibly share datum illustrate model producer consumer problem representative paradigm operate system function specifically section 3.5 describe bound buffer enable process share return consideration bound buffer point original solution allow buffer size 1 item buffer time suppose want modify algorithm remedy deficiency possibility add integer variable count initialize 0 count incremente time add new item buffer decremente time remove item buffer code producer process modify follow true produce item produce count = = buffer size buffer[in = produce = + 1 buffer size code consumer process modify follows true count = = 0 consume = buffer[out = + 1 buffer size consume item consume producer consumer routine show correct separately function correctly execute concurrently illustration suppose value variable count currently 5 producer consumer process concurrently execute statement count++ count-- follow execution statement value variable count 4 5 6 correct result count = = 5 generate correctly producer consumer value count incorrect follow note statement count++ implement machine language typical machine follow register1 = count register1 = register1 + 1 count = register1 register1 local cpu register similarly statement count- implement follow register2 = count register2 = register2 1 count = register2 register2 local cpu register register1 register2 physical register remember content register save restore interrupt handler section 1.2.3 concurrent execution count++ count--

equivalent sequential execution low level statement present previously interleave arbitrary order order high level statement preserve interleaving follow register1 = count register1 = 5 register1 = register1 + 1 register1 = 6 register2 = count register2 = 5 register2 = register2 1 register2 = 4 count = register1 count = 6 count = register2 count = 4

notice arrive incorrect state count = = 4 indicate buffer fact buffer reverse order statement t4 t5 arrive incorrect state count arrive incorrect state allow process manipulate variable count concurrently situation like process access manipulate datum concurrently outcome execution depend particular order access take place call race condition guard race condition need ensure process time manipulate variable count guarantee require process synchronize way situation describe occur frequently operate system different part system manipulate resource furthermore emphasize early chapter prominence multicore sys- tem bring increase emphasis develop multithreaded appli- cation application thread possibly shar- e datum run parallel different processing core clearly want change result activity interfere importance issue devote major portion chapter process synchronization coordination cooperate critical section problem begin consideration process synchronization discuss so- call critical section problem consider system consist n process p0 p1 pn1 process segment code call critical section process access update datum share process important feature system process execute critical section process allow execute critical section process execute critical section time critical section problem design protocol process use synchronize activity cooperatively share datum process request permission enter critical section section code implement request entry section critical section follow exit section remain code remainder section general structure typical process show figure 6.1 entry section exit section enclose box highlight important segment code solution critical section problem satisfy follow 1 mutual exclusion process pi execute critical section process execute critical section 2 progress process execute critical section pro- cesse wish enter critical section process execute remainder section participate decid- ing enter critical section selection true general structure typical process critical section problem 3 bound waiting exist bound limit number time process allow enter critical section process request enter critical section request grant assume process execute nonzero speed assumption concern relative speed n process give point time kernel

assumption concern relative speed n process give point time kernel mode process active operate system result code implement operate system kernel code subject possible race condition consider example kernel datum structure maintain list open file system list modify new file open close add file list remove list process open file simultaneously separate update list result race condition example illustrate figure 6.2 situation pro- cesse p0 p1 create child process fork system recall section 3.3.1 fork return process identifier newly create process parent process example race condi- tion variable kernel variable available pid represent value available process identifier mutual exclusion provide possible process identifier number assign separate process kernel datum structure prone possible race condition include structure maintain memory allocation maintain process list interrupt handling kernel developer ensure operate system free race condition critical section problem solve simply single core envi- ronment prevent interrupt occur share variable modify way sure current sequence next_available_pid = 2615 pid_t child = fork child = 2615 child = 2615 pid_t child = fork race condition assign pid instruction allow execute order preemption instruction run unexpected modification share variable unfortunately solution feasible multiprocessor environ- ment disable interrupt multiprocessor time consume message pass processor message passing delay entry critical section system efficiency decrease consider effect system clock clock keep update interrupt general approach handle critical section operate system preemptive kernel nonpreemptive kernel preemptive ker- nel allow process preempt run kernel mode nonpreemptive kernel allow process run kernel mode preempt kernel mode process run exit kernel mode block voluntarily yield control cpu obviously nonpreemptive kernel essentially free race condition kernel datum structure process active kernel time preemptive kernel carefully design ensure share kernel datum free race condition pre- emptive kernel especially difficult design smp architecture environment possible kernel mode process run simultaneously different cpu core favor preemptive kernel

mode process run simultaneously different cpu core favor preemptive kernel nonpreemp- tive preemptive kernel responsive risk kernel mode process run arbitrarily long period relin- quishe processor wait process course risk minimize design kernel code behave way fur- thermore preemptive kernel suitable real time programming allow real time process preempt process currently run illustrate classic software base solution critical section prob- lem know peterson solution way modern computer architecture perform basic machine language instruction load store guarantee peterson solution work correctly architecture present solution provide good algorithmic description solve critical section problem illus- trate complexity involve design software address requirement mutual exclusion progress bound waiting peterson solution restrict process alternate execution critical section remainder section process num- bered p0 p1 convenience present pi use pj denote process j equal 1 i peterson solution require process share datum item true flag[i = true turn = j flag[j turn = = j critical section flag[i = false /*remainder section structure process pi peterson solution variable turn indicate turn enter critical section turn = = process pi allow execute critical section flag array indicate process ready enter critical section example flag[i true pi ready enter critical section explanation datum structure complete ready describe algorithm show figure 6.3 enter critical section process pi set flag[i true set turn value j assert process wish enter critical section process try enter time turn set j roughly time assignment occur overwrite imme- diately eventual value turn determine process allow enter critical section prove solution correct need 1 mutual exclusion preserve 2 progress requirement satisfy 3 bound waiting requirement meet prove property 1 note pi enter critical section flag[j = = false turn = = i. note process execute critical section time flag[0 = = flag[1 = = true observation imply p0 p1 successfully execute statement time value turn 0 1 process pj successfully execute statement pi execute additional statement turn = = j time flag[j = = true turn = = j condition persist long pj critical section result mutual exclusion prove property 2 3 note process pi prevent enter critical section stick loop condition flag[j = = true turn = = j loop possible pj ready enter critical section flag[j = = false pi enter critical section pj set flag[j true execute statement turn = = turn = = j. turn = = pi enter critical section turn = = j pj enter critical section pj exit critical section reset flag[j false allow pi enter critical section pj reset flag[j true set turn i. pi

change value variable turn execute statement pi enter critical section progress entry pj bound waiting mention beginning section peterson solution guarantee work modern computer architecture primary rea- son improve system performance processor and/or compiler reorder read write operation dependency single- threaded application reordering immaterial far program correct- ness concern final value consistent expect similar balance checkbook actual order credit debit operation perform unimportant final balance multithreaded application share datum reordering instruction render inconsistent unexpected result example consider following datum share boolean flag = false int x = 0 thread 1 perform statement thread 2 perform x = 100 flag = true expect behavior course thread 1 output value 100 variable x. data dependency variable flag x possible processor reorder instruction thread 2 flag assign true assignment x = 100 situation possible thread 1 output 0 variable x. obvious processor reorder statement issue thread 1 load variable x load value flag occur thread 1 output 0 variable x instruction issue thread 2 reorder hardware support synchronization flag[0 = true turn = 1 turn = 0

flag[1 = true effect instruction reordering peterson solution affect peterson solution consider happen assignment statement appear entry section peterson solution figure 6.3 reorder possible thread active critical section time show figure 6.4 follow section way preserve mutual exclusion proper synchronization tool discussion tool begin primitive support hardware proceed abstract high level software base api available kernel developer application programmer hardware support synchronization describe software base solution critical section prob- lem refer software base solution algorithm involve special support operate system specific hardware instruction ensure mutual exclusion discuss software base solution guarantee work modern computer architecture section present hardware instruction provide support solve critical section problem primitive operation directly synchronization tool form foundation abstract synchronization mechanism section 6.3 see system reorder instruction policy lead unreliable data state computer architecture determine memory guarantee provide application program know memory model general memory model fall category 1 strongly order memory modification processor immediately visible processor 2 weakly order modification memory processor immediately visible processor memory model vary processor type kernel developer assumption visibility modification memory share memory multiprocessor address issue computer architecture provide instruction force change memory propagate processor ensure memory modification visible thread run processor instruction know memory barrier memory fence memory barrier instruction perform system ensure load store complete subse- quent load store operation perform instruction reorder memory barrier ensure store operation com- plete memory visible processor future load store operation perform let return recent example reordering instruction result wrong output use memory barrier ensure obtain expect output add memory barrier operation thread 1 guarantee value flag load value x.

similarly place memory barrier assignment per- form thread 2 x = 100 flag = true ensure assignment x occur assignment flag respect peterson solution place memory barrier assignment statement entry section avoid reordering operation show figure 6.4 note memory barrier consider low level operation typically kernel developer write specialized code ensure mutual exclusion modern computer system provide special hardware instruction allow test modify content word swap content word atomically uninterruptible unit use special instruction solve critical section problem relatively simple manner discuss specific instruction specific machine abstract main concept type instruction describe test set compare swap instruction boolean test set(boolean target boolean rv = target target = true definition atomic test set instruction hardware support synchronization test set(&lock critical section lock = false remainder section true mutual exclusion implementation test set test set instruction define show figure 6.5 important characteristic instruction execute atomi- cally test set instruction execute simultaneously different core execute sequentially arbitrary order machine support test set instruction implement mutual exclusion declare boolean variable lock initialize false structure process pi show figure 6.6 compare swap instruction cas like test set instruction operate word atomically use different mechanism base swap content word cas instruction operate operand define figure 6.7 operand value set new value expression value = = expect true regardless cas return original value variable value important characteristic instruction execute atomically cas instruction execute simultaneously different core execute sequentially arbitrary mutual exclusion cas provide follow global vari- able lock declare initialize 0 process invoke compare swap set lock 1 enter critical section int compare swap(int value int expect int new value int temp = value value = = expect value = new value definition atomic compare swap instruction true compare swap(&lock 0 1 = 0

critical section lock = 0 remainder section mutual exclusion compare swap instruction original value lock equal expect value 0 subse- quent call compare swap succeed lock equal expect value 0 process exit critical section set lock 0 allow process enter critical section structure process pi show figure 6.8 algorithm satisfy mutual exclusion requirement satisfy bound waiting requirement figure 6.9 present true waiting[i = true key = 1 waiting[i key = = 1 key = compare swap(&lock,0,1 waiting[i = false critical section j = + 1 n j = waiting[j j = j + 1 n j = = lock = 0 waiting[j = false remainder section bound waiting mutual exclusion compare swap hardware support synchronization make compare swap atomic intel x86 architecture assembly language statement cmpxchg implement compare swap instruction enforce atomic execution lock prefix lock bus destination operand update general form instruction appear lock cmpxchg < destination operand > < source operand > algorithm compare swap instruction satisfy critical section requirement common data structure element wait array initialize false lock initial- ize 0 prove mutual exclusion requirement meet note process pi enter critical section waiting[i = = false key = = 0 value key 0 compare swap execute process execute compare swap find key = = 0 wait variable waiting[i false process leave critical section waiting[i set false maintain mutual exclusion requirement prove progress requirement meet note argument present mutual exclusion apply process exit critical section set lock 0 set waiting[j false allow process wait enter critical section proceed prove bound waiting requirement meet note process leave critical section scan array wait cyclic ordering + 1 + 2 n 1 0 1 designate process ordering entry section waiting[j = = true enter critical section process wait enter critical section n 1 turn detail describe implementation atomic test set

compare swap instruction discuss fully book com- typically compare swap instruction directly provide mutual exclusion basic building block construct tool solve critical section problem tool atomic variable provide atomic operation basic datum type integer boolean know section 6.1

incremente decremente integer value produce race condition atomic variable ensure mutual exclusion situation data race single variable update counter incremente system support atomic variable provide special atomic datum type function access manipulate atomic variable function implement compare swap opera- tion example follow increment atomic integer sequence increment function implement cas instruction void increment(atomic int v temp = v temp = compare swap(v temp temp+1 important note atomic variable provide atomic update entirely solve race condition circumstance example bound buffer problem describe section 6.1 use atomic integer count ensure update count atomic producer consumer process loop condition depend value count consider situation buffer currently consumer loop wait count > 0 producer enter item buffer consumer exit loop count long equal 0

proceed consume value count set 1 atomic variable commonly operate system con- current application use limit single update share datum counter sequence generator follow sec- tion explore robust tool address race condition gen- hardware base solution critical section problem present sec- tion 6.4 complicated generally inaccessible application pro- grammer instead operating system designer build high level software tool solve critical section problem simple tool mutex lock fact term mutex short mutual exclusion use mutex lock protect critical section prevent race condition process acquire lock enter critical section release lock exit critical section acquire()function acquire lock release function release lock illustrate figure 6.10 mutex lock boolean variable available value indicate lock available lock available acquire succeed lock consider unavailable process attempt acquire unavailable lock block lock release true solution critical section problem mutex lock definition acquire follow busy wait available = false definition release follow available = true call acquire release perform atomically mutex lock implement cas operation describe section 6.4 leave description technique exercise lock contended uncontended lock consider contended thread block try acquire lock lock available thread attempt acquire lock consider uncontended con- tended lock experience high contention relatively large number thread attempt acquire lock low contention relatively small number thread attempt acquire lock unsurprisingly highly contend lock tend decrease overall performance concurrent mean short duration spinlock identify locking mechanism choice multi- processor system lock hold short duration exactly constitute short duration give wait lock require context switch context switch thread waiting state second context switch restore wait thread lock available general rule use spinlock lock hold duration context switch main disadvantage implementation give require busy waiting process critical section process try enter critical section loop continuously acquire continual looping clearly problem real multiprogramme system single cpu core share process busy waiting waste cpu cycle process able use productively

process busy waiting waste cpu cycle process able use productively section 6.6 examine strategy avoid busy waiting temporarily put wait process sleep awaken lock type mutex lock describe call spin- lock process spin wait lock avail- able issue code example illustrate com- pare swap instruction spinlock advantage context switch require process wait lock context switch considerable time certain circumstance multi- core system spinlock fact preferable choice locking lock hold short duration thread spin process core thread perform critical section core modern multicore compute system spinlock widely operate chapter 7 examine mutex lock solve classical synchronization problem discuss mutex lock spinlock operate system pthread mutex lock mention early generally consider simple synchronization tool section examine robust tool behave similarly mutex lock provide sophisticated way process synchronize activity semaphore s integer variable apart initialization access standard atomic operation wait signal semaphore introduce dutch computer scientist edsger dijk- stra wait operation originally term p dutch proberen test signal originally call v verhogen incre- ment definition wait follow s < = 0

busy wait definition signal follow modification integer value semaphore wait signal operation execute atomically process modify semaphore value process simultaneously modify semaphore value addition case wait(s testing integer value s s 0 possible modification s-- execute interruption shall operation implement section 6.6.2 let semaphore semaphore value count semaphore range unrestricted domain value binary semaphore range 0 1 binary semaphore behave similarly mutex lock fact system provide mutex lock binary semaphore instead provide mutual exclusion count semaphore control access give resource consist finite number instance semaphore initialize number resource available process wish use resource perform wait operation semaphore decremente count process release resource perform signal operation incremente count count semaphore go 0 resource process wish use resource block count great 0 use semaphore solve synchronization problem example consider concurrently run process p1 statement s1 p2 statement s2 suppose require s2 execute s1 complete implement scheme readily let p1 p2 share common semaphore synch initialize 0 process p1 insert process p2 insert statement synch initialize 0 p2

execute s2 p1 invoke signal(synch statement s1 execute recall implementation mutex lock discuss section 6.5 suffer busy waiting definition wait signal semaphore operation describe present problem overcome prob- lem modify definition wait signal operation follow process execute wait operation find semaphore value positive wait engage busy waiting process suspend suspend operation place process waiting queue associate semaphore state process switch wait state control transfer cpu scheduler select process execute aprocess suspend wait semaphore s restart process execute signal operation process restart wakeup operation change process waiting state ready state process place ready queue cpu switch run process newly ready process depend cpu scheduling algorithm implement semaphore definition define semaphore typedef struct struct process list semaphore integer value list process list process wait semaphore add list process signal operation remove process list wait process awaken process wait semaphore operation define wait(semaphore s s->value < 0 add process s->list signal semaphore operation define signal(semaphore s s->value < = 0 remove process p s->list sleep operation suspend process invoke wakeup(p operation resume execution suspend process p. opera- tion provide operate system basic system call note implementation semaphore value negative semaphore value negative classical definition semaphore busy waiting semaphore value negative magnitude number process wait semaphore fact result switch order decrement test implementation list wait process easily implement link field process control block pcb semaphore contain integer value pointer list pcb way add remove process list ensure bounded waiting use fifo queue semaphore contain head tail pointer queue general list use queuing strategy correct usage semaphore depend particular queuing strategy semaphore list mention critical semaphore operation execute atomi- cally guarantee process execute wait sig- nal operation semaphore time critical- section problem single processor environment solve sim- ply inhibit interrupt time wait signal operation execute scheme work single processor environment interrupt inhibit instruction different process interleave currently run process execute interrupt reenable scheduler regain control multicore environment interrupt disable pro- cesse core instruction different process run dif- ferent core interleave arbitrary way disable interrupt core difficult task seriously diminish performance smp system provide alternative technique com- pare swap spinlock ensure wait signal per- important admit completely eliminate busy waiting definition wait signal operation move busy wait entry section critical section application program furthermore limit busy waiting critical section wait signal operation section short properly code instruction critical section occupy busy waiting occur rarely short time entirely different situation exist application program critical section long minute hour occupy case busy waiting extremely inefficient semaphore provide convenient effective mechanism pro- cess synchronization incorrectly result timing error difficult detect error happen particular execution sequence place sequence occur see example error use count solution producer consumer problem section 6.1 example timing problem happen rarely count value appear reasonable 1 solution obviously acceptable reason mutex lock semaphore appear reasonable 1 solution obviously acceptable reason mutex lock semaphore introduce place unfortunately timing error occur mutex lock semaphore illustrate review semaphore solution critical section problem process share binary semaphore variable mutex initialize 1 process execute wait(mutex enter critical section signal(mutex afterward sequence observe process critical section simultaneously list difficulty result note difficulty arise single process behave situation cause honest programming error uncooperative programmer suppose program interchange order wait signal operation semaphore mutex execute result follow execution situation process execute critical sec- tion simultaneously violate mutual exclusion requirement error discover process simultaneously active critical section note situation repro- suppose program replace signal(mutex wait(mutex execute case process permanently block second wait semaphore unavailable suppose process omit wait(mutex signal(mutex case mutual exclusion violate process example illustrate type error generate easily programmer use semaphore mutex lock incorrectly solve critical section problem strategy deal error incor- porate simple synchronization tool high level language construct section describe fundamental high level synchronization construct monitor type abstract datum type adt encapsulate datum set function operate datum independent specific implementation adt monitor type adt include set programmer define operation provide mutual exclusion monitor monitor type declare variable value define state monitor monitor share variable declaration function p1 function p2 function pn initialization code pseudocode syntax monitor instance type body function operate variable syntax monitor type show figure 6.11 repre- sentation monitor type directly process function define monitor access variable declare locally monitor formal parameter similarly local variable monitor access local function monitor construct ensure process time active monitor consequently programmer need code synchronization constraint explicitly figure 6.12 monitor construct define far sufficiently powerful model synchronization scheme purpose need define additional syn- chronization mechanism mechanism provide condition construct programmer need write tailor synchronization scheme define variable type condition condition x y operation invoke condition variable wait signal operation mean

x y operation invoke condition variable wait signal operation mean process invoke operation suspend schematic view monitor x.signal operation resume exactly suspend process process suspend signal operation effect state x operation execute figure 6.13 contrast operation signal operation associate semaphore affect state semaphore suppose x.signal operation invoke pro- cess p exist suspend process q associate condition x. clearly suspend process q allow resume execution signaling process p wait p q active simultane- ously monitor note conceptually process continue execution possibility exist 1 signal wait p wait q leave monitor wait 2 signal continue q wait p leave monitor wait condition reasonable argument favor adopt option hand p execute monitor signal and- continue method reasonable allow thread p continue time q resume logical condition q wait long hold compromise choice exist thread p execute signal operation immediately leave monitor q immediately resume queue associate x y condition monitor condition variable programming language incorporate idea monitor describe section include java c language erlang provide concurrency support similar mechanism implement monitor semaphore consider possible implementation monitor mechanism semaphore monitor binary semaphore mutex initialize 1 provide ensure mutual exclusion process execute wait(mutex enter monitor execute signal(mutex leave use signal wait scheme implementation signaling process wait resume process leave wait additional binary semaphore introduce initialize 0 signal process use suspend integer variable count provide count number process suspend external function f replace body f count > 0 mutual exclusion monitor ensure describe condition variable implement condition x introduce binary semaphore x sem integer variable x count initialize 0 operation x.wait count > 0 operation x.signal implement x count > 0

implementation applicable definition monitor give hoare brinch hansen bibliographical note end chapter case generality implementation void acquire(int time busy = true void release busy = false initialization code busy = false monitor allocate single resource unnecessary significant improvement efficiency possible leave problem exercise 6.27 resume process monitor turn subject process resumption order monitor process suspend condition x x.signal opera- tion execute process determine suspend process resume simple solution use come serve fcfs ordering process wait- e long resume circumstance simple scheduling scheme adequate circumstance conditional- wait construct construct form c integer expression evaluate wait operation execute value c call priority number store process suspend x.signal execute process small priority number resume illustrate new mechanism consider resourceallocator mon- itor show figure 6.14 control allocation single resource compete process process request allocation resource specify maximum time plan use resource mon- itor allocate resource process short time allocation request process need access resource question observe following sequence access resource r instance type resourceallocator unfortunately monitor concept guarantee precede access sequence observe particular following problem process access resource gain access permission resource process release resource grant access resource process attempt release resource request process request resource twice release difficulty encounter use semaphore difficulty similar nature encourage develop monitor construct place previously worry correct use semaphore worry correct use high level programmer define operation compiler long assist possible solution current problem include resource- access operation resourceallocator monitor solution mean scheduling accord build monitor scheduling algorithm code ensure process observe appropriate sequence inspect program use resourceallocator monitor manage resource check condition establish correct- ness system user process call monitor correct sequence second sure uncooperative process simply ignore mutual exclusion gateway provide monitor try access share resource directly access protocol condition ensure guarantee time dependent error occur scheduling algorithm inspection possible small static system reasonable large system scheduling algorithm inspection possible small static system reasonable large system dynamic system access control problem solve use additional mechanism describe chapter 17 consequence synchronization tool coordinate access critical section possibility process attempt enter critical section wait indefinitely recall section 6.2 outline criterion solution critical section problem satisfy indefinite waiting violate progress bound waiting criterion liveness refer set property system satisfy ensure process progress execution life cycle process wait- e indefinitely circumstance describe example different form liveness failure gen- erally characterize poor performance responsiveness simple example liveness failure infinite loop busy wait loop present possibility liveness failure especially process loop arbitrarily long period time effort provide mutual exclusion tool mutex lock semaphore lead failure concurrent pro- gramming section explore situation lead liveness implementation

semaphore wait queue result situation process wait indefinitely event cause wait process event question execution signal operation state reach process say deadlocke illustrate consider system consist process p0 p1 access semaphore s q set value 1 suppose p0 execute wait(s p1 execute wait(q p0 execute wait(q wait p1 execute signal(q similarly p1 execute wait(s wait p0 execute signal(s signal operation execute p0 p1 deadlocked set process deadlocked state process set wait event cause process set event mainly concern acquisition release resource mutex lock semaphore type event result deadlock detail chapter 8 chapter describe mechanism deal deadlock problem form liveness failure scheduling challenge arise high priority process need read modify kernel datum currently access low priority process chain low priority process kernel datum typi- cally protect lock high priority process wait low priority finish resource situation complicated low priority process preempt favor process high priority example assume process l m h priority follow order l < m < h. assume process h require semaphore s currently access process l. ordinarily process h wait l finish resource s. suppose process m runnable preempt process l. indirectly process low priority process m affect long process h wait l relinquish resource s.

liveness problem know priority inversion occur system priority typically priority inversion avoid implement priority inheritance protocol accord protocol process access resource need high priority process inherit high priority finish resource question finish priority revert original value example priority inheritance protocol allow process l temporarily inherit priority process h prevent process m preempt execution process l finish resource s relinquish inherit priority h assume original priority resource s available process h m run describe different synchronization tool solve critical section problem give correct implementation usage tool effectively ensure mutual exclusion address liveness issue growth concurrent program leverage power modern multicore computer system increase attention pay performance synchronization tool try identify use tool daunting challenge section present simple strategy determine use specific synchro- hardware solution outline section 6.4 consider low level typically foundation construct synchro- nization tool mutex lock recent focus cas instruction construct lock free algorithm provide protection race condition require overhead locking lock free solution gain popularity low overhead priority inversion mars pathfinder priority inversion scheduling inconvenience system tight time constraint real time system priority inversion cause process long accomplish task happen failure cascade result system failure consider mars pathfinder nasa space probe land robot sojourner rover mars 1997 conduct experiment shortly sojourner begin operate start experience frequent computer reset reset reinitialize hardware software include communica- tion problem solve sojourner fail mission problem cause fact high priority task bc dist take long expect complete work task force wait share resource hold low priority asi met task turn preempt multiple medium priority task bc dist task stall wait share resource ultimately bc sched task discover problem perform reset sojourner suffer typical case priority inversion operate system sojourner vxworks real time operat- e system global variable enable priority inheritance semaphore testing variable set sojourner mars problem solve ability scale algorithm difficult develop set sojourner mars problem solve ability scale algorithm difficult develop test exercise end chapter ask evaluate correctness lock free stack cas base approach consider optimistic approach opti- mistically update variable use collision detection thread update variable concurrently repeatedly retry operation successfully update conflict mutual- exclusion locking contrast consider pessimistic strategy assume thread concurrently update variable pessimistically acquire lock make update following guideline identify general rule concern performance difference cas base synchronization traditional synchroniza- tion mutex lock semaphore vary contention load uncontended option generally fast cas protection somewhat fast traditional synchronization moderate contention cas protection fast possibly fast traditional synchronization high contention highly contend load traditional synchro- nization ultimately fast cas base synchronization moderate contention particularly interesting examine scenario cas operation succeed time fail iterate loop show figure 6.8 time ultimately suc- ceede comparison mutual exclusion locking attempt acquire contend lock result complicated time intensive code path suspend thread place wait queue require context switch thread choice mechanism address race condition greatly affect system performance example atomic integer light weight traditional lock generally appropriate mutex lock semaphore single update share variable counter design operate system spinlock multiprocessor system lock hold short duration general mutex lock simple require overhead semaphore preferable binary semaphore protect access critical section use control access finite number resource count semaphore generally appropriate mutex lock similarly instance reader writer lock prefer mutex lock allow high degree concurrency multiple appeal high level tool monitor condition variable base simplicity ease use tool significant overhead depend implementation likely scale highly contend situation fortunately ongoing research develop scalable efficient tool address demand concurrent programming design compiler generate efficient code develop language provide support concurrent programming improve performance exist library api chapter examine operate system api available developer implement synchronization tool present race condition occur process concurrent access share datum final result depend condition occur process concurrent access share datum final result depend particular order con- current access occur race condition result corrupt value critical section section code shared datum manipu- late possible race condition occur critical section problem design protocol process synchronize activity cooperatively share datum solution critical section problem satisfy following requirement 1 mutual exclusion 2 progress 3 bound waiting mutual exclusion ensure process time active crit- ical section progress ensure program cooperatively determine process enter critical section bound waiting limit time program wait enter critical section software solution critical section problem peterson solu- tion work modern computer architecture hardware support critical section problem include memory barri- er hardware instruction compare swap instruction mutex lock provide mutual exclusion require process acquire lock enter critical section release lock exit critical section semaphore like mutex lock provide mutual exclusion mutex lock binary value indicate lock available semaphore integer value solve variety synchronization problem monitor abstract data type provide high level form process synchronization monitor use condition variable allow process wait certain condition true signal condition set true solution critical section problem suffer liveness prob- lem include deadlock tool solve critical section problem synchronize activity process evaluate vary level contention tool work well certain con- tention load section 6.4 mention disable interrupt frequently affect system clock explain occur effect minimize meaning term busy waiting kind waiting operate system busy waiting avoid altogether explain answer explain spinlock appropriate single processor system multiprocessor system wait signal semaphore operation execute atomically mutual exclusion violate illustrate binary semaphore implement mutual exclusion n process race condition possible computer system consider banking system maintain account balance function deposit(amount withdraw(amount function pass deposit withdraw bank account balance assume husband wife share bank account concurrently husband call withdraw function wife call deposit describe race condition possible prevent race condition occur mutual exclusion describe race condition possible prevent race condition occur mutual exclusion problem discuss classic paper dijk- stra 1965 semaphore concept suggest dijkstra 1965 monitor concept develop brinch hansen 1973 hoare 1974 give complete description monitor mars pathfinder problem http://research.microsoft.co

m en um people mbj mars pathfinder authoritative account.html thorough discussion memory barrier cache memory present mckenney 2010 herlihy shavit 2012 present detail issue relate multiprocessor programming include memory model compare swap instruction bahra 2013 examine nonblocking algo- rithm modern multicore system s. a. bahra nonblocking algorithm scalable multicore programming acm queue volume 11 number 5 2013 p. brinch hansen operating system principles prentice e. w. dijkstra cooperate sequential process technical report technological university eindhoven netherlands 1965 herlihy shavit 2012 m. herlihy n. shavit art multiprocessor programming revised edition morgan kaufmann publishers inc. 2012 c. a. r. hoare monitor operating system structuring concept communications acm volume 17 number 10 1974 page p. e. mckenney memory barrier hardware view software hackers 2010 chapter 6 exercise pseudocode figure 6.15 illustrate basic push pop operation array base stack assume algorithm concurrent environment answer following question datum race condition race condition fix race condition possible computer system consider online auction system current high bid item maintain person wish bid item call bid(amount function compare bid current high bid exceed current high bid high bid set new illustrate void bid(double > highestbid highestbid = < size stack[top = item = = 0 array base stack exercise 6.12 sum array series partial sum exercise 6.14 describe race condition possible situation prevent race condition occur follow program example sum array value size n element parallel system contain n compute core separate processor array element j = 1 log 2(n k = 1 n k + 1 pow(2,j = = 0 values[k + = values[k pow(2,(j-1 effect sum element array series partial sum show figure 6.16 code execute sum element array store array location race condition code example identify occur

illustrate example demonstrate algorithm free race condition compare swap instruction design lock free data structure stack queue list program example show figure 6.17 present possible solution lock free stack cas instruction stack represent link list node element represent stack implementation free race condition typedef struct node value t datum struct node node stack void push(value t item node old node node new node new node = malloc(sizeof(node new node->data = item old node = new node->next = old node compare swap(top old node new node = old node value t pop node old node node new node old node = old node = = null new node = old node->next compare swap(top old node new node = old node return old node->data lock free stack exercise 6.15 approach compare swap implement spin- lock follow void lock spinlock(int lock compare swap(lock 0 1 = 0 spin suggest alternative approach use compare compare- swap idiom check status lock invoke compare swap operation rationale approach invoke compare swap()only lock currently available strategy show void lock spinlock(int lock true lock = = 0 lock appear available compare swap(lock 0 1 compare compare swap idiom work appropriately implement spinlock explain illustrate integrity lock compromise semaphore implementation provide function getvalue return current value semaphore function instance invoke prior call wait process wait value semaphore > 0 prevent blocking wait semaphore example getvalue(&sem > 0 developer argue function discourage use describe potential problem occur function getvalue scenario know correct software solution critical section problem process develop dekker process p0 p1 share following variable boolean flag[2 initially false structure process pi = = 0 1 show figure 6.18 process pj j = = 1 0

prove algorithm satisfy requirement critical section problem know correct software solution critical section problem n process low bound waiting n 1 turn present eisenberg mcguire process share following enum pstate idle want cs true flag[i = true flag[j turn = = j flag[i = false turn = = j flag[i = true critical section turn = j flag[i = false remainder section structure process pi dekker algorithm element flag initially idle initial value turn immaterial 0 n-1 structure process pi show figure 6.19 prove algorithm satisfy requirement critical section problem explain implement synchronization primitive disable interrupt appropriate single processor system synchro- nization primitive user level program pare swap instruction assume following structure define mutex lock available typedef struct value available = = 0 indicate lock available value 1 indicate lock unavailable struct illustrate following function implement compare swap instruction void acquire(lock mutex void release(lock mutex sure include initialization necessary true true flag[i = want j = turn j = flag[j = idle j = turn j = j + 1 n flag[i = cs j = 0 j < n j = = || flag[j = cs j > = n turn = = || flag[turn = = idle critical section j = turn + 1 n flag[j = = idle j = j + 1 n

turn = j flag[i = idle remainder section structure process pi eisenberg mcguire algorithm explain interrupt appropriate implement synchro- nization primitive multiprocessor system implementation mutex lock provide section 6.5 suffer busy waiting describe change necessary process wait acquire mutex lock block place waiting queue lock available assume system multiple processing core follow scenario describe well locking mechanism spinlock mutex lock waiting process sleep wait lock available lock hold short duration lock hold long duration thread sleep hold lock assume context switch take t time suggest upper bound term t hold spinlock spinlock hold long mutex lock wait thread sleep well multithreaded web server wish track number request service know hit consider follow strate- gy prevent race condition variable hit strategy use basic mutex lock update hit mutex lock hit lock second strategy use atomic integer atomic t hit explain strategy efficient consider code example allocate release process show figure 6.20 identify race condition(s assume mutex lock name mutex operation acquire release indicate locking need place prevent race condition(s replace integer variable int number process = 0 atomic integer atomic t number process = 0 prevent race condition(s server design limit number open connection example server wish n socket connection point time soon n connection server accept incoming connection existing connection define max process 255 int number process = 0

implementation fork call function int allocate process int new pid number process = = max process allocate necessary process resource + + number process return new pid implementation exit call function void release process release process resource --number process allocate release process exercise 6.27 release illustrate semaphore server limit number concurrent connection section 6.7 use follow illustration incorrect use semaphore solve critical section problem explain example liveness failure demonstrate monitor semaphore equivalent degree implement solution type syn- describe signal operation associate monitor differ corresponding operation define semaphore suppose signal statement appear statement monitor function suggest implementation describe section 6.7 simplify situation consider system consist process p1 p2 pn unique priority number write monitor allocate identical printer process priority number decide order allocation file share different process unique number file access simultaneously process subject following constraint sum unique number associate process currently access file n.

write monitor coordinate access file signal perform condition inside monitor signal process continue execution transfer control process signal solution precede exercise differ different way signaling perform design algorithm monitor implement alarm clock enable call program delay specified number time unit tick assume existence real hardware clock invoke function tick monitor regular interval discuss way priority inversion problem address real time system discuss solution implement context proportional share assume finite number resource single resource type manage process ask number resource return finish example commercial software package provide give number license indicate number application run concurrently application start license count decremente application terminate license count incremente license use request start application deny request grant existing license holder terminate application license following program segment manage finite number instance available resource maximum number resource number available resource declare follow define max resource 5 int available resource = max resource process wish obtain number resource invoke decrease count function decrease available resource count resource return 0 sufficient resource available return -1 int decrease count(int count available resource < count available resource -= count process want return number resource call increase count function increase available resource count int increase count(int count available resource + = count precede program segment produce race condition fol- identify datum involve race condition identify location location code race con- semaphore mutex lock fix race condition permissible modify decrease count function calling process block sufficient resource available decrease count function previous exercise currently return 0 sufficient resource available 1 lead awkward programming process wish obtain number resource decrease count(count = = -1

rewrite resource manager code segment monitor con- dition variable decrease count function suspend process sufficient resource available allow process invoke decrease count simply call process return function sufficient resource available c h p t e r chapter 6 present critical section problem focus race condition occur multiple concurrent process share datum go examine tool address critical section problem prevent race condition occur tool range low level hardware solution memory barrier compare swap oper- ation increasingly high level tool mutex lock semaphore monitor discuss challenge design application free race condition include liveness hazard deadlock chapter apply tool present chapter 6 classic synchronization problem explore synchronization mechanism linux unix windows operate system describe api detail java posix system explain bound buffer reader writer dining philosopher describe specific tool linux windows solve process illustrate posix java solve process synchroniza- design develop solution process synchronization problem posix java api classic problem synchronization section present number synchronization problem example large class concurrency control problem problem test nearly newly propose synchronization scheme solution problem use semaphore synchronization true produce item produce add produce buffer structure producer process traditional way present solution actual implementation solution use mutex lock place binary semaphore bound buffer problem bound buffer problem introduce section 6.1 commonly illustrate power synchronization primitive present gen- eral structure scheme commit particular implementation provide related programming project exercise end chapter problem producer consumer process share following semaphore mutex = 1 semaphore = n semaphore = 0 assume pool consist n buffer capable hold item mutex binary semaphore provide mutual exclusion access buffer pool initialize value 1 semaphore count number buffer semaphore initialize value n semaphore initialize value 0

code producer process show figure 7.1 code consumer process show figure 7.2 note symmetry producer consumer interpret code producer produce buffer consumer consumer produce buffer producer readers writers problem suppose database share concurrent process process want read database want update read write database distinguish classic problem synchronization true remove item buffer consume consume item consume structure consumer process type process refer reader writer obviously reader access share datum simultaneously adverse effect result writer process reader writer access database simultaneously chaos ensue ensure difficulty arise require writer exclusive access share database write database synchronization problem refer reader writer problem originally state test nearly new synchronization reader writer problem variation involve priori- tie simple refer reader writer problem require reader keep wait writer obtain permission use share object word reader wait read- er finish simply writer wait second reader writer problem require writer ready writer perform write soon possible word writer wait access object new reader start read solution problem result starvation case writer starve second case reader starve reason variant problem propose present solution reader writer problem bibliographical note end chapter reference describe starvation free solution second solution reader writer problem reader process share following data structure semaphore rw mutex = 1 semaphore mutex = 1 int read count = 0 binary semaphore mutex rw mutex initialize 1 read count counting semaphore initialize 0

semaphore rw mutex true writing perform structure writer process common reader writer process mutex semaphore ensure mutual exclusion variable read count update read count variable keep track process currently read object semaphore rw mutex function mutual exclusion semaphore writer reader enter exit critical section reader enter exit reader critical section code writer process show figure 7.3 code reader process show figure 7.4 note writer critical section n reader wait reader queue rw mutex n 1 reader queue mutex observe writer execute sig- nal(rw mutex resume execution wait reader single wait writer selection scheduler reader writer problem solution generalize provide reader writer lock system acquire reader writer lock require specify mode lock read write access true read count = = 1 reading perform read count = = 0 structure reader process classic problem synchronization process wish read share datum request reader writer lock read mode process wish modify share datum request lock write mode multiple process permit concurrently acquire reader writer lock read mode process acquire lock write exclusive access require writer reader writer lock useful following situation application easy identify process read share datum process write share datum application reader writer reader writer lock generally require overhead establish semaphore mutual exclusion lock increase concurrency allow multiple reader compensate overhead involve set reader writer lock dining philosopher problem consider philosopher spend life think eat philosopher share circular table surround chair belong philosopher center table bowl rice table lay single chopstick figure 7.5

philosopher think interact colleague time time philosopher get hungry try pick chopstick close chopstick left right neighbor philosopher pick chopstick time obviously pick chopstick hand neighbor hungry philosopher chopstick time eat release chopstick finish eat put chopstick start think dining philosopher problem consider classic synchronization problem practical importance computer scientist dislike philosopher example large class concurrency control problem simple representation need situation dining philosopher true wait(chopstick[(i+1 5 eat signal(chopstick[(i+1 5 think awhile structure philosopher i. allocate resource process deadlock free simple solution represent chopstick semaphore philosopher try grab chopstick execute wait operation semaphore release chopstick execute signal operation appropriate semaphore share datum element chopstick initialize 1 structure philosopher show figure 7.6 solution guarantee neighbor eat simul- taneously reject create deadlock suppose philosopher hungry time grab left chopstick element chopstick equal 0 philosopher try grab right chopstick delay possible remedy deadlock problem follow allow philosopher sit simultaneously table allow philosopher pick chopstick chopstick available pick critical section use asymmetric solution odd number philosopher pick left chopstick right chopstick even- number philosopher pick right chopstick left section 6.7 present solution dining philosopher problem ensure freedom deadlock note satisfactory solution dining philosopher problem guard possibility synchronization kernel philosopher starve death deadlock free solution necessarily eliminate possibility starvation illustrate monitor concept present deadlock free solution dining philosopher problem solution impose restriction philosopher pick chopstick available code solution need distinguish state find philosopher purpose introduce following data structure enum thinking hungry eating state[5 philosopher set variable state[i = eating neigh- bor eat state[(i+4 5 = eating state[(i+1 5 = eating need declare allow philosopher delay hungry unable obtain chopstick need position describe solution dining- philosopher problem distribution chopstick control monitor diningphilosophers definition show figure 7.7

philosopher start eat invoke operation pickup act result suspension philosopher process successful completion operation philosopher eat follow philosopher invoke putdown operation philosopher invoke operation pickup putdown follow easy solution ensure neighbor eat simultaneously deadlock occur note possible philosopher starve death present solution problem leave exercise synchronization kernel describe synchronization mechanism provide windows linux operate system operate system provide good example different approach synchronize kernel enum thinking hungry eating state[5 void pickup(int state[i = hungry state[i = eating void putdown(int state[i = thinking test((i + 4 5 test((i + 1 5 void test(int state[(i + 4 5 = eating state[i = = hungry state[(i + 1 5 = eating state[i = eating initialization code int = 0 < 5 i++ state[i =

thinking monitor solution dining philosopher problem synchronization mechanism available system differ subtle significant way synchronization windows windows operate system multithreaded kernel provide sup- port real time application multiple processor windows kernel access global resource single processor system temporar- ily mask interrupt interrupt handler access global resource multiprocessor system windows protect access global resource spinlock kernel use spinlock protect short code segment furthermore reason efficiency kernel ensure thread preempt hold spinlock synchronization kernel thread synchronization outside kernel windows provide dis- patcher object dispatcher object thread synchronize accord different mechanism include mutex lock semaphore event timer system protect shared datum require thread gain owner- ship mutex access datum release ownership finish semaphore behave describe section 6.6 event similar condition variable notify wait thread desire condition occur finally timer notify thread specified time expire dispatcher object signal state nonsignaled state object signal state available thread block acquire object object nonsignaled state available thread block attempt acquire object illustrate state transition mutex lock dispatcher object figure 7.8 relationship exist state dispatcher object state thread thread block nonsignaled dispatcher object state change ready wait thread place wait queue object state dispatcher object move signal kernel check thread wait object kernel move thread possibly wait state ready state resume execute number thread kernel select wait queue depend type dispatcher object thread wait kernel select thread wait queue mutex mutex object own single thread event object kernel select thread wait event use mutex lock illustration dispatcher object thread state thread try acquire mutex dispatcher object nonsignaled state thread suspend place wait queue mutex object mutex move signal state thread release lock mutex thread wait queue move wait state ready state acquire mutex lock critical section object user mode mutex acquire release kernel intervention multiprocessor system critical section object use spinlock wait thread release object spin long acquire

object use spinlock wait thread release object spin long acquire thread allocate kernel mutex yield cpu critical section object particularly efficient kernel mutex allocate contention object practice little contention saving significant owner thread release mutex lock thread acquire mutex lock mutex dispatcher object provide programming project end chapter use mutex lock semaphore windows api synchronization linux prior version 2.6

linux nonpreemptive kernel mean process run kernel mode preempt high priority process available run linux kernel fully preemptive task preempt run kernel linux provide different mechanism synchronization kernel computer architecture provide instruction atomic version simple math operation simple synchronization technique linux kernel atomic integer represent opaque datum type atomic t. imply math operation atomic integer perform interruption illustrate consider program consist atomic integer counter integer value atomic t counter follow code illustrate effect perform atomic opera- counter = 5 counter = counter + 10 counter = counter 4 counter = counter + 1 value = atomic read(&counter value = 12

atomic integer particularly efficient situation integer variable counter need update atomic operation require overhead locking mechanism use limit sort scenario situation variable contribute possible race condition sophisticated locking tool mutex lock available linux protect critical section kernel task invoke mutex lock function prior enter critical section mutex unlock function exit critical section mutex lock unavailable task call mutex lock sleep state awaken lock owner invoke mutex unlock linux provide spinlock semaphore reader writer version lock lock kernel smp machine fun- damental locking mechanism spinlock kernel design spinlock hold short duration single processor machine embedded system single processing core spinlock inappropriate use replace enable disable kernel pre- emption system single processing core hold spinlock kernel disable kernel preemption release spinlock enable kernel preemption summarize acquire spin lock release spin lock disable kernel preemption enable kernel preemption linux kernel spinlock mutex lock nonrecursive mean thread acquire lock acquire lock second time release lock second attempt acquire lock block linux use interesting approach disable enable kernel preemp- tion provide simple system call preempt disable pre- empt enable()—for disable enable kernel preemption kernel preemptible task run kernel hold lock enforce rule task system thread info structure contain- e counter preempt count indicate number lock hold task lock acquire preempt count incremente decre- mente lock release value preempt count task currently run kernel great 0 safe preempt ker- nel task currently hold lock count 0

kernel safely interrupt assume outstanding call preempt disable spinlock enable disable kernel preemption kernel lock disable kernel preemption hold short duration lock hold long period semaphore mutex lock appropriate use synchronization method discuss precede section pertain synchronization kernel available kernel developer contrast posix api available programmer user level particular operate system kernel course ultimately implement tool provide host operating section cover mutex lock semaphore condition variable available pthreads posix api api widely thread creation synchronization developer unix linux posix mutex locks mutex lock represent fundamental synchronization technique pthreads mutex lock protect critical section code thread acquire lock enter critical section release exit critical section pthreads use pthread mutex t data type mutex lock mutex create pthread mutex init function parameter pointer mutex pass null second parameter initialize mutex default attribute illustrate pthread mutex t mutex create initialize mutex lock pthread mutex init(&mutex null mutex acquire release pthread mutex lock pthread mutex unlock function mutex lock unavailable pthread mutex lock invoke call thread block owner invoke pthread mutex unlock following code illustrate pro- tecte critical section mutex lock acquire mutex lock pthread mutex lock(&mutex critical section release mutex lock pthread mutex unlock(&mutex mutex function return value 0

correct operation error occur function return nonzero error code system implement pthreads provide semaphore semaphore posix standard instead belong posix sem extension posix specify type semaphore name unnamed fundamentally similar differ term create share process technique common discuss begin version 2.6 kernel linux system provide support named unnamed semaphore posix name semaphore function sem open create open posix name sempahore sem t sem create semaphore initialize 1 sem = sem open("sem o creat 0666 1 instance name semaphore sem o creat flag indicate semaphore create exist additionally semaphore read write access process parameter 0666 initialize 1 advantage name semaphore multiple unrelated process easily use common semaphore synchronization mechanism simply refer semaphore example semaphore sem create subsequent call sem open parameter process return descriptor exist section 6.6 describe classic wait signal semaphore operation posix declare operation sem wait sem post respectively following code sample illustrate protect critical section name semaphore create acquire semaphore critical section release semaphore linux macos system provide posix name semaphore posix unnamed semaphores unnamed semaphore create initialize sem init func- tion pass parameter 1 pointer semaphore 2 flag indicate level sharing 3 semaphore initial value illusrate following programming example sem t sem create semaphore initialize 1 sem init(&sem 0 1 example pass flag 0 indicate semaphore share thread belong process create semaphore supply nonzero value allow semaphore share separate process place region share memory addition initialize semaphore value 1 posix unnamed semaphore use sem wait sem post operation name semaphore follow code sample illustrate pro- tecte critical section unnamed semaphore create acquire semaphore critical section release semaphore like mutex lock semaphore function return 0 successful nonzero error condition occur posix condition variable condition variable pthreads behave similarly describe section 6.7 section condition variable context monitor provide locking mechanism ensure data integrity pthreads typically c program c monitor accomplish locking associate condition variable condition variable pthreads use pthread cond t data type initialize pthread cond init function following code create initialize condition variable associate mutex lock pthread mutex t mutex pthread cond t cond var pthread mutex init(&mutex null pthread cond init(&cond var null pthread cond wait function wait condition variable following code illustrate thread wait condition = = b true pthread condition variable pthread mutex lock(&mutex = b pthread cond wait(&cond var mutex pthread mutex unlock(&mutex mutex lock associate condition variable lock pthread cond wait function call protect datum conditional clause possible race condition lock acquire thread check condition condition true thread invoke pthread cond wait pass mutex lock condition variable parameter call pthread cond wait release mutex lock allow thread access share datum possibly update value condition clause evaluate true protect program error important place conditional clause loop condition rechecke signal synchronization java athread modify share datum invoke pthread cond signal function signal thread wait condition variable illustrate pthread mutex lock(&mutex = b pthread cond signal(&cond var pthread mutex unlock(&mutex important note pthread cond signal release mutex lock subsequent pthread mutex unlock release mutex mutex lock release signal thread owner mutex lock return control pthread cond wait provide programming problem project end chapter use pthreads mutex lock condition variable posix synchronization java java language api provide rich support thread syn- chronization origin language section cover java monitor java original synchronization mechanism cover additional mechanism introduce release 1.5 reentrant lock semaphore condition variable include represent common locking synchronization mechanism java api provide feature cover text exam- ple support atomic variable cas instruction encourage interested text exam- ple support atomic variable cas instruction encourage interested reader consult bibliography information java provide monitor like concurrency mechanism thread synchroniza- tion illustrate mechanism boundedbuffer class figure 7.9 implement solution bound buffer problem pro- ducer consumer invoke insert remove method respec- object java associate single lock method declare synchronize call method require own lock object declare synchronize method place synchronized keyword method definition insert remove method boundedbuffer class invoke synchronize method require own lock object instance boundedbuffer lock own thread thread call synchronize method block place entry set object lock entry set represent set thread wait lock available lock available synchronize method call call thread owner object lock enter method lock release thread exit method entry set lock lock release jvm public class boundedbuffer < e >

private static final int buffer size = 5 private int count private e buffer public boundedbuffer count = 0 = 0 = 0 buffer = e

new object[buffer size producer method public synchronized void insert(e item figure 7.11 consumer method public synchronize e remove figure 7.11 bound buffer java synchronization arbitrarily select thread set owner lock arbitrarily mean specification require thread set organize particular order practice virtual machine order thread entry set accord fifo policy figure 7.10 illustrate entry set operate addition have lock object associate wait set consist set thread wait set initially thread enter synchronize method own lock object thread determine unable continue certain condition entry set lock synchronization java time lock acquire release define scope lock synchronize method small percentage code manipulate share datum yield scope large instance well synchronize block code manipulate share datum synchronize entire method design result small lock scope addition declare synchronize method java allow block synchronization illustrate access critical section code require ownership object lock object public void somemethod non critical section critical section remainder section meet happen example producer call insert method buffer thread release lock wait condition allow continue meet thread call wait method following happen 1 thread release lock object 2 state thread set block 3 thread place wait set object consider example figure 7.11 producer call insert method see buffer call wait method release lock block producer put producer wait set object producer release lock consumer ultimately enter remove method free space buffer producer figure 7.12 illustrate entry wait set lock note wait throw interruptedexception choose ignore code clarity simplicity consumer thread signal producer proceed ordinarily thread exit synchronize method depart thread release lock associate object possibly remove thread entry set give ownership lock end insert remove method method notify notify 1 pick arbitrary thread t list thread wait set producer method public synchronize void insert(e item count = = buffer size catch interruptedexception ie buffer[in = item = + 1 buffer size consumer

ie buffer[in = item = + 1 buffer size consumer method public synchronized e remove count = = 0 catch interruptedexception ie item = buffer[out = + 1 buffer size figure 7.11 insert remove method wait notify 2 move t wait set entry set 3 set state t block runnable t eligible compete lock thread t regain control lock return call wait check value count selection arbitrary thread accord java specification practice java virtual machine order thread wait set accord fifo policy synchronization java entry wait set describe wait notify method term method show figure 7.11 assume buffer lock object available producer call insert method see lock available enter method method producer determine buffer call wait wait release lock object set state producer block put producer wait set object consumer ultimately call enter remove method lock object available consumer remove item buffer call notify note consumer own lock object notify remove producer wait set object move producer entry set set producer state consumer exit remove method exit method release lock object producer try reacquire lock successful resume execu- tion wait producer test loop determine room available buffer proceed remainder insert method thread wait set object notify ignore producer exit method release lock object synchronized wait notify mechanism java origin later revision java api introduce flexible robust locking mechanism examine follow section simple locking mechanism available api reentrant- lock way reentrantlock act like synchronized statement describe section 7.4.1 reentrantlock own single thread provide mutually exclusive access shared resource reentrantlock provide additional feature set fairness parameter favor grant lock long wait thread recall specification jvm indicate thread wait set object lock order specific fashion thread acquire reentrantlock lock invoke lock method lock available thread invoke lock own term reentrant lock assign invoke thread lock ownership return control lock unavailable invoke thread block

thread lock ownership return control lock unavailable invoke thread block ultimately assign lock owner invoke unlock reentrantlock implementsthe lock interface follow lock key = new reentrantlock critical section programming idiom try finally require bit expla- nation lock acquire lock method important lock similarly release enclose unlock finally clause ensure lock release critical section complete excep- tion occur try block notice place lock try clause lock throw check exception con- sider happen place lock try clause unchecked exception occur lock invoke outofmemoryerror finally clause trigger unlock throw unchecked illegalmonitorstateexception lock acquire ille- galmonitorstateexception replace unchecked exception occur lock invoke obscure reason program reentrantlock provide mutual exclusion con- servative strategy multiple thread read write share datum describe scenario section 7.1.2 address need java api provide reentrantreadwritelock lock allow multiple concurrent reader writer java api provide count semaphore describe section 6.6 constructor semaphore appear value specify initial value semaphore negative value allow acquire method throw interruptedexception acquire thread interrupt follow example illustrate semaphore mutual exclusion synchronization java semaphore sem = new semaphore(1 critical section catch interruptedexception ie notice place release finally clause ensure semaphore release utility cover java api condition variable reentrantlock similar java synchronized statement condition variable provide functionality similar wait notify method provide mutual exclusion condition variable associate reentrant lock create condition variable create reentrantlock invoke newcondition method return condition object rep- resent condition variable associate reentrantlock illustrate following statement lock key = new reentrantlock condition condvar = key.newcondition condition variable obtain invoke await signal method function way wait signal command describe section 6.7 recall monitor describe section 6.7 wait signal operation apply name condition variable allow thread wait specific condition notify specific condition meet language level java provide support name condition variable java monitor associate unnamed condition variable wait notify operation describe section 7.4.1 apply single condition variable java thread

operation describe section 7.4.1 apply single condition variable java thread awaken notify receive information awaken reactivate thread check condition wait meet condition variable remedy allow specific thread notify illustrate follow example suppose thread number 0 4 share variable turn indicate thread turn thread wish work call dowork method figure 7.13 pass thread number thread value threadnumber match value turn proceed thread wait threadnumber thread wish work public void dowork(int threadnumber turn wait signal threadnumber = turn work awhile signal thread turn = turn + 1 5

catch interruptedexception ie example java condition variable create reentrantlock condition variable repre- sente condition thread wait signal thread turn show lock lock = new reentrantlock condition condvar = new condition[5 int = 0 < 5 i++ condvars[i = lock.newcondition thread enter dowork

invoke await method associate condition variable threadnumber equal turn resume signal thread thread complete work signal condition variable associate thread turn important note dowork need declare syn- chronize reentrantlock provide mutual exclusion thread invoke await condition variable release associate reen- trantlock allow thread acquire mutual exclusion lock similarly signal invoke condition variable signal lock release invoke unlock emergence multicore system come increase pressure develop concurrent application advantage multiple processing core concurrent application present increase risk race condition liveness hazard deadlock traditionally technique mutex lock semaphore monitor address issue number processing core increase increasingly difficult design multithreaded application free race condition deadlock section explore feature provide programming language hardware support design thread safe computer science idea area study solve problem area concept transactional memory orig- inate database theory example provide strategy process synchronization memory transaction sequence memory read write operation atomic operation transaction complete memory transaction commit operation abort roll benefit transactional memory obtain feature add programming language consider example suppose function update modify share datum traditionally function write mutex lock semaphore follow void update modify share datum synchronization mechanism mutex lock additionally number thread increase traditional locking scale level contention thread lock ownership high alternative traditional locking method new feature advantage transactional memory add programming language example suppose add construct atomic{s ensure operation s execute transaction allow rewrite update function follow void update modify share datum advantage mechanism lock transactional memory system developer responsible guaranteeing atomicity additionally lock involve deadlock possible furthermore transactional memory system identify statement atomic block execute concurrently concurrent read access share variable course possible programmer identify situation use reader writer lock task increasingly difficult number thread application grow transactional memory implement software hardware software transactional memory stm suggest implement transactional memory exclusively software special hardware need stm work insert instrumentation code inside transaction block

hardware need stm work insert instrumentation code inside transaction block code insert compiler manage transaction examine statement run concurrently specific low level locking

require hardware transactional memory htm use hardware cache hierar- chie cache coherency protocol manage resolve conflict involve share datum reside separate processor cache htm require special code instrumentation overhead stm htm require exist cache hierarchy cache coherency protocol modify support transactional memory transactional memory exist year widespread implementation growth multicore system associ- ate emphasis concurrent parallel programming prompt significant research area academic commercial software hardware vendor section 4.5.2 provide overview openmp support parallel programming share memory environment recall openmp include set compiler directive api code follow compiler direc- tive pragma omp parallel identify parallel region perform number thread equal number processing core system advantage openmp similar tool thread creation man- agement handle openmp library responsibility pragma omp parallel compiler directive openmp pro- vide compiler directive pragma omp critical specify code region follow directive critical section thread active time way openmp provide support ensure thread generate race condition example use critical section compiler directive assume shared variable counter modify update function follow void update(int value counter + = value update function invoke parallel region race condition possible variable counter critical section compiler directive remedy race condition code follow void update(int value pragma omp critical counter + = value critical section compiler directive behave like binary semaphore mutex lock ensure thread time active critical section thread attempt enter critical section thread currently active section own section call thread block owner thread exit multiple critical section critical section assign separate rule specify thread active critical section advantage critical section compiler directive openmp generally consider easy use standard mutex lock disadvantage application developer identify possible race condition adequately protect share datum compiler directive additionally critical section compiler directive behave like mutex lock deadlock possible critical section identify functional programming language know programming language c c++ java c know imperative procedural language imperative language implement

c++ java c know imperative procedural language imperative language implement algorithm state base language flow algorithm crucial correct operation state represent variable data structure course program state mutable variable assign different value time current emphasis concurrent parallel programming multicore system great focus functional programming language follow programming paradigm different offer imperative language fundamental difference imper- ative functional language functional language maintain state variable define assign value value immutable change functional language disallow muta- ble state need concern issue race condition deadlock essentially problem address chapter nonexistent functional language functional language presently use briefly mention erlang scala erlang language gain significant attention support concurrency ease develop application run parallel system scala func- tional language object orient fact syntax scala similar popular object orient language java c reader inter- este erlang scala detail functional language general encourage consult bibliography end chapter additional reference classic problem process synchronization include bound buffer reader writer dining philosopher problem solution problem develop tool present chapter 6 includ- ing mutex lock semaphore monitor condition variable windows use dispatcher object event implement process linux use variety approach protect race condition include atomic variable spinlock mutex lock posix api provide mutex lock semaphore condition variable posix provide form semaphore name unnamed unrelated process easily access name semaphore sim- ply refer unnamed semaphore share easily require place semaphore region share memory java rich library api synchronization available tool include monitor provide language level reentrant lock semaphore condition variable support alternative approach solve critical section problem include transactional memory openmp functional language functional lan- guage particularly intriguing offer different programming paradigm procedural language unlike procedural language func- tional language maintain state generally immune race condition critical section explain windows linux implement multiple lock mech- anism describe circumstance use spinlock mutex lock semaphore condition variable case explain mechanism need windows provide lightweight

semaphore condition variable case explain mechanism need windows provide lightweight synchronization tool call slim reader writer lock implementation reader writer lock favor reader writer order

waiting thread fifo policy slim reader writer lock favor reader writer wait thread order fifo queue explain benefit provide synchronization tool describe change necessary producer con- sumer process figure 7.1 figure 7.2 mutex lock instead binary semaphore describe deadlock possible dining philosopher prob- explain difference signal non signaled state windows dispatcher object assume val atomic integer linux system value val follow operation complete detail windows synchronization find solomon russi- novich 2000 love 2010 describe synchronization linux kernel hart 2005 describe thread synchronization windows breshears 2009 pacheco 2011 provide detailed coverage synchronization issue relation parallel programming detail openmp find http://openmp.org oaks 2014 goetz et al 2006 con- trast traditional synchronization cas base strategy java c. breshears art concurrency o'reilly associates goetz et al 2006 b. goetz t. peirls j. bloch j. bowbeer d. holmes d. lea java concurrency practice addison wesley 2006 j. m. hart windows system programming edition addison- r. love linux kernel development edition developer s. oaks java performance definitive guide o'reilly asso- p. s. pacheco introduction parallel programming morgan solomon russinovich 2000 d. a. solomon m. e. russinovich inside microsoft windows 2000 edition microsoft press 2000 chapter 7 exercise

describe kernel data structure race condition possi- ble sure include description race condition occur linux kernel policy process hold spinlock attempt acquire semaphore explain policy place design algorithm bound buffer monitor buffer portion embed monitor strict mutual exclusion monitor make bound buffer monitor exercise 7.14 mainly suitable small portion explain true design new scheme suitable large portion discuss tradeoff fairness throughput operation reader writer problem propose method solve reader writer problem cause starvation explain lock method java reentrantlock place try clause exception handling unlock method place finally clause explain difference software hardware transactional exercise 3.20 require design pid manager allocate unique process identifier process exercise 4.28 require modify solution exercise 3.20 write program create number thread request release process identifier mutex lock modify solution exercise 4.28 ensure data structure represent availability process identifier safe race condition exercise 4.27 write program generate fibonacci sequence program require parent thread wait child thread finish execution print compute value let parent thread access fibonacci number soon compute child thread wait child thread terminate change necessary solution exercise implement modify solution c program stack ptr.c available source code download contain implementation stack link list example use follow stacknode = null int value = pop(&top value = pop(&top value = pop(&top program currently race condition appropriate concurrent environment pthreads mutex lock describe section 7.3.1 fix race condition exercise 4.24 ask design multithreaded program esti- mate π monte carlo technique exercise ask create single thread generate random point store result global variable thread exit parent thread perform calculation estimate value π modify pro- gram create thread generate random point determine point fall circle thread update global count point fall circle protect race condition update share global variable mutex lock exercise 4.25 ask design program openmp esti- mate π monte carlo technique examine solution program look possible race

π monte carlo technique examine solution program look possible race condition identify race condition protect strategy outline section 7.5.2 barrier tool synchronize activity number thread thread reach barrier point proceed thread reach point thread reach barrier point thread release resume concurrent assume barrier initialize n number thread wait barrier point thread perform work reach barrier point work awhile work awhile posix java synchronization tool describe chapter construct barrier implement following api int init(int n)—initialize barrier specify size int barrier point(void)—identifies thread release barrier thread reach return value function identify error condition function return 0 normal operation return 1 error occur testing harness provide source code download test implementation barrier project 1 design thread pool thread pool introduce section 4.5.1 thread pool task submit pool execute thread pool work submit pool queue available thread remove work queue available thread work remain queued available work thread await notification task available project involve create manage thread pool complete pthreds posix synchronization java provide detail relevant specific technology posix version project involve create number thread pthreads api posix mutex lock semaphore user thread pool utilize following api void pool init()—initialize thread pool int pool submit(void somefunction)(void p void p somefunction pointer function execute thread pool p parameter pass function void pool shutdown(void)—shuts thread pool task provide example program client.c source code download illustrate use thread pool function implementation thread pool source code download provide c source file

threadpool.c partial implementation thread pool need implement function call client user additional function support internal thread pool implementation involve 1 pool init function create thread startup initialize mutual exclusion lock semaphore 2 pool submit function partially implement currently place function execute datum task struct task struct represent work complete thread pool pool submit add task queue invok- e enqueue function worker thread dequeue retrieve work queue queue implement statically array dynamically link list pool init function int return value indicate task successfully submit pool 0 indicate success 1 indicate failure queue implement array pool init return 1 attempt submit work queue queue implement link list pool init return 0 memory allocation error occur 3 worker function execute thread pool thread wait available work work available thread remove queue invoke execute run semaphore notify wait thread work submit thread pool named unnamed semaphore refer section 7.3.2 detail posix 4 mutex lock necessary avoid race condition access modify queue section 7.3.1 provide detail pthreads mutex 5 pool shutdown function cancel worker thread wait thread terminate call pthread join refer section 4.6.3 detail posix thread cancellation semaphore operation sem wait cancellation point allow thread wait semaphore cancel refer source code download additional detail project particular readme file describe source header file makefile build project java version project complete java synchroniza- tion tool describe section 7.4 synchronization depend monitor synchronized wait()/notify section 7.4.1 b semaphore reentrant lock section 7.4.2 section 7.4.3 java thread describe section 4.4.3 implementation thread pool thread pool implement following api threadpool()—create default sized thread pool threadpool(int size)—create thread pool size size void add(runnable task)—add task perform thread void shutdown()—stop thread pool provide java source file

threadpool.java partial implemen- tation thread pool source code download need imple- ment method call client user additional method support internal thread pool implementation involve following activity 1 constructor create number idle thread await work 2 work submit pool add method add task implement runnable interface add method place runnable task queue use available structure java api java.util list 3 thread pool available work check queue runnable task task idle thread remove task queue invoke run method queue idle thread wait notify work available add method implement notification notify semaphore operation place runnable task queue possibly awaken idle thread await work 4 shutdown method stop thread pool invoke interrupt method course require runnable task execute thread pool check interruption status section refer source code download additional detail project particular readme file describe java source file detail java thread interruption project 2 sleep teaching assistant university computer science department teaching assistant ta help undergraduate student programming assignment regular office hour ta office small room desk chair computer chair hallway outside office student sit wait ta currently help student student need help office hour ta sit desk take nap student arrive office hour find ta sleep student awaken ta ask help student arrive find ta currently help student student sit chair hallway wait chair available student come later time posix thread mutex lock semaphore implement solu- tion coordinate activity ta student detail assignment provide student ta pthread section 4.4.1 begin create n student student run separate thread ta run separate thread student thread alternate program period time seek help ta tai available obtain help sit chair hallway chair available resume program seek help later time student arrive notice ta sleep student notify ta semaphore ta finish help student ta check student wait help hallway ta help student turn student present ta return good

hallway ta help student turn student present ta return good option simulate student programming ta provide help student appropriate thread sleep random period time coverage posix mutex lock semaphore provide section 7.3 consult section detail project 3 dining philosopher problem section 7.1.3 provide outline solution dining philosopher problem monitor project involve implement solution problem posix mutex lock condition variable java condition variable solution base algorithm illustrate implementation require create philosopher identi- fie number 0 4 philosopher run separate thread philoso- pher alternate think eating simulate activity thread sleep random period second thread creation pthreads cover section 4.4.1 philosopher wish eat invoke function pickup forks(int philosopher number philosopher number identify number philosopher wish eat philosopher finish eat invoke return forks(int philosopher number implementation require use posix condition variable cover section 7.3

philosopher wish eat invoke method take- forks(philosophernumber philosophernumber number philosopher wish eat philosopher finish eat invoke returnforks(philosophernumber solution implement following interface public interface diningserver call philosopher wish eat public void takeforks(int philosophernumber call philosopher finish eat public void returnforks(int philosophernumber require use java condition variable cover section project 4 producer consumer problem section 7.1.1 present semaphore base solution producer consumer problem bound buffer project design programming solution bound buffer problem producer consumer process show figure 5.9 5.10 solution present section 7.1.1 use semaphore count number slot buffer mutex binary mutual- exclusion semaphore protect actual insertion removal item buffer project use standard count semaphore mutex lock binary semaphore represent mutex producer consumer run separate thread item buffer synchronize mutex structure solve problem pthreads windows api internally buffer consist fix size array type buffer item define typedef array buffer item object manipulate circular queue definition buffer item size buffer store header file follow buffer.h typedef int buffer item define buffer size 5 buffer manipulate function insert item remove item call producer consumer thread respectively skeleton outline function appear figure 7.14 insert item remove item function synchronize producer consumer algorithm outline figure 7.1 figure 7.2 buffer require initialization function initialize mutual exclusion object mutex semaphore main function initialize buffer create separate pro- ducer consumer thread create producer consumer thread main function sleep period time awaken- e terminate application main function pass parameter command line 1 long sleep terminate 2 number producer thread 3 number consumer thread skeleton function appear figure 7.15 buffer buffer item buffer[buffer size int insert item(buffer item item insert item buffer return 0 successful return -1 indicate error condition int remove item(buffer item item remove object buffer place item return 0 successful

return -1 indicate error condition outline buffer operation producer consumer thread producer thread alternate sleep random period time insert random integer buffer random number produce rand function produce random integer 0 rand max consumer sleep random period time awakening attempt remove item buffer outline producer consumer thread appear figure 7.16 int main(int argc char argv 1 command line argument argv[1],argv[2],argv[3 2 initialize buffer 3 create producer thread(s 4 create consumer thread(s 5 sleep 6 exit outline skeleton program include < stdlib.h >

require rand void producer(void param buffer item item true sleep random period time generate random number item = rand insert item(item fprintf("report error condition printf("producer produce dn",item void consumer(void param buffer item item true sleep random period time remove item(&item fprintf("report error condition printf("consumer consume dn",item outline producer consumer thread note early solve problem pthreads windows api follow section supply information choice pthreads thread creation synchronization create thread pthreads api discuss section 4.4.1 coverage mutex lock semaphore pthreads provide section 7.3 refer section specific instruction pthread thread creation section 4.4.2 discuss thread creation windows api refer section specific instruction create thread windows mutex locks mutex lock type dispatcher object describe section 7.2.1 follow illustrate create mutex lock createmutex mutex = createmutex(null false null parameter refer security attribute mutex lock set attribute null prevent child process create mutex lock inherit handle lock second parameter indicate creator mutex lock lock initial owner pass value false indicate thread create mutex initial owner shall soon mutex lock acquire parameter allow mutex provide value null mutex successful createmutex return handle mutex lock return null section 7.2.1 identify dispatcher object signal nonsignaled signal dispatcher object mutex lock available ownership acquire move nonsignaled state release return signal mutex lock acquire invoke waitforsingleobject func- tion function pass handle lock flag indicate long wait follow code demonstrate mutex lock create acquire parameter value infinite indicate wait infinite time lock available value allow call thread time lock available specify time lock signal state waitforsingleobject return immediately lock nonsignaled lock release move signal state invoke releasemutex()—for example semaphore windows api dispatcher object use signaling mechanism mutex lock semaphore create follow sem = createsemaphore(null 1 5 null parameter identify security attribute semaphore similar describe mutex lock second parameter indicate initial value maximum value semaphore instance initial value semaphore 1 maximum value 5

value semaphore instance initial value semaphore 1 maximum value 5 successful createsemaphore return handle mutex lock return null semaphore acquire waitforsingleobject func- tion mutex lock acquire semaphore sem create example following statement value semaphore > 0

semaphore signal state acquire call thread call thread block indefinitely specify infinite semaphore return signal state equivalent signal operation windows semaphore releasesemaphore function function pass parameter 1 handle semaphore 2 increase value semaphore 3 pointer previous value semaphore use follow statement increase sem 1 releasesemaphore(sem 1 null releasesemaphore releasemutex return nonzero value successful 0 c h p t e r multiprogramme environment thread compete finite number resource thread request resource resource available time thread enter waiting state wait thread change state resource request hold wait thread situation call deadlock discuss issue briefly chapter 6 form liveness failure define deadlock situation process set process wait event cause process set good illustration deadlock draw law pass kansas legislature

early 20th century say train approach crossing shall come stop shall start go chapter describe method application developer operating system programmer use prevent deal dead- lock application identify program dead- lock operate system typically provide deadlock prevention facil- itie remain responsibility programmer ensure design deadlock free program deadlock problem liveness failure challenging demand continue increase concurrency parallelism multicore system illustrate deadlock occur mutex lock define necessary condition characterize deadlock identify deadlock situation resource allocation graph evaluate different approach prevent deadlock apply banker algorithm deadlock avoidance apply deadlock detection algorithm evaluate approach recover deadlock system consist finite number resource distribute number compete thread resource partition type class consist number identical instance cpu cycle file o device network interface dvd drive example resource type system cpu resource type cpu instance similarly resource type network instance thread request instance resource type allocation instance type satisfy request instance identical resource type class define synchronization tool discuss chapter 6 mutex lock semaphore system resource contemporary com- puter system common source deadlock def- inition problem lock typically associate specific data structure lock protect access queue protect access link list forth reason instance lock typically assign resource class note chapter discuss kernel resource thread use resource process example interprocess commu- nication resource use result deadlock deadlock concern kernel describe thread request resource release resource thread request resource require carry designate task obviously number resource request exceed total number resource available system word thread request network interface system normal mode operation thread utilize resource following sequence 1 request thread request resource request grant immediately example mutex lock currently hold thread request thread wait acquire 2 use thread operate resource example resource mutex lock thread access critical section 3 release thread release resource request release resource system call explain chapter 2 example request release device open close file allocate

chapter 2 example request release device open close file allocate free memory system call similarly see chapter 6 request release accomplish wait signal operation semaphore acquire release mutex lock use kernel manage resource thread operate system check sure thread request allocate resource system table record resource free allocate resource allocate deadlock multithreaded applications table record thread allocate thread request resource currently allocate thread add queue thread wait resource aset thread deadlocked state thread set wait- e event cause thread set event mainly concern resource acquisition release resource typically logical example mutex lock semaphore file type event result deadlock include read- e network interface ipc interprocess communication facility discuss chapter 3 illustrate deadlocked state refer dining philosopher problem section 7.1.3 situation resource represent chopstick philosopher hungry time philosopher grab chopstick left long available chopstick philosopher block wait right chopstick developer multithreaded application remain aware possibility deadlock locking tool present chapter 6 design avoid race condition tool developer pay careful attention

lock acquire release deadlock occur describe deadlock multithreaded application prior examine deadlock issue identify man- age illustrate deadlock occur multithreaded pthread program posix mutex lock pthread mutex init function initialize unlock mutex mutex lock acquire pthread mutex lock pthread mutex unlock respectively thread attempt acquire lock mutex pthread mutex lock block thread owner mutex lock invoke pthread mutex unlock mutex lock create initialize following code example pthread mutex t mutex pthread mutex t second mutex pthread mutex init(&first mutex null pthread mutex init(&second mutex null thread thread thread create thread access mutex lock thread thread run function work work respectively

show figure 8.1 example thread attempt acquire mutex lock order 1 mutex 2 second mutex time thread attempt acquire mutex lock order 1 second mutex 2 mutex deadlock possible thread acquire mutex thread acquire second mutex thread run function void work one(void param pthread mutex lock(&first mutex pthread mutex lock(&second mutex work pthread mutex unlock(&second mutex pthread mutex unlock(&first mutex thread run function void work two(void param pthread mutex lock(&second mutex pthread mutex lock(&first mutex work pthread mutex unlock(&first mutex pthread mutex unlock(&second mutex note deadlock possible occur thread acquire release mutex lock mutex second mutex thread attempt acquire lock course order thread run depend schedule cpu scheduler example illustrate problem handle deadlock difficult identify test deadlock occur certain livelock form liveness failure similar deadlock prevent thread proceed thread unable proceed different reason deadlock occur thread set block wait event cause thread set livelock occur thread continuously attempt action fail livelock similar happen people attempt pass hallway move right left obstruct progress move left move right forth block make livelock illustrate pthreads pthread mutex trylock function attempt acquire mutex lock block code example figure 8.2 rewrite example figure 8.1 use pthread mutex trylock situation lead livelock thread acquire mutex follow thread acquire second mutex thread invoke pthread mutex trylock fail release respective lock repeat action indefinitely livelock typically occur thread retry fail operation time generally avoid have thread retry fail operation random time precisely approach take ethernet network network collision occur try retransmit packet immediately collision occur host involve collision backoff random period time attempt transmit livelock common deadlock nonetheless challenge issue design concurrent application like deadlock occur specific scheduling circumstance previous section illustrate deadlock occur multi- threaded programming mutex lock look closely con- dition characterize deadlock deadlock situation arise follow condition hold simultane- ously system 1 mutual exclusion resource

follow condition hold simultane- ously system 1 mutual exclusion resource hold nonsharable mode thread time use resource thread request resource request thread delay resource release 2 hold wait thread hold resource wait acquire additional resource currently hold 3 preemption resource preempt resource release voluntarily thread hold thread complete task 4 circular wait aset t0 t1 tn wait thread exist t0 wait resource hold t1 t1 wait resource hold t2 tn1 wait resource hold tn tn wait resource hold t0 emphasize condition hold deadlock occur circular wait condition imply hold wait condition thread run function void work one(void param int = 0

pthread mutex lock(&first mutex pthread mutex trylock(&second mutex work pthread mutex unlock(&second mutex pthread mutex unlock(&first mutex = 1 pthread mutex unlock(&first mutex thread run function void work two(void param int = 0 pthread mutex lock(&second

mutex pthread mutex trylock(&first mutex work pthread mutex unlock(&first mutex pthread mutex unlock(&second mutex = 1 pthread mutex unlock(&second mutex resource allocation graph program figure 8.1 condition completely independent shall section 8.5 how- useful consider condition separately deadlock describe precisely term direct graph call system resource allocation graph graph consist set vertex v set edge e. set vertex v partition different type node t = t1 t2 tn set consist active thread system r = r1 r2 rm set consist resource type direct edge thread ti resource type rj denote ti rj signify thread ti request instance resource type rj currently wait resource direct edge resource type rj thread ti denote rj ti signify instance resource type rj allocate thread ti direct edge ti rj call request edge direct edge rj ti call assignment edge pictorially represent thread ti circle resource type rj rectangle simple example resource allocation graph show figure 8.3 illustrate deadlock situation program figure 8.1 resource type rj instance represent instance dot rectangle note request edge point rectangle rj assignment edge designate dot rectangle thread ti request instance resource type rj request edge insert resource allocation graph request fulfil request edge instantaneously transform assignment edge thread long need access resource release resource result assignment edge delete resource allocation graph show figure 8.4 depict follow set t r e t = t1 t2 t3 r = r1 r2 r3 r4 e = t1 r1 t2 r3 r1 t2 r2 t2 r2 t1 r3 t3

resource instance instance resource type r1 instance resource type r2 instance resource type r3 instance resource type r4 thread state thread t1 hold instance resource type r2 wait instance resource type r1 thread t2 hold instance r1 instance r2 wait instance r3 thread t3 hold instance r3 give definition resource allocation graph show graph contain cycle thread system deadlocke graph contain cycle deadlock exist resource type exactly instance cycle imply deadlock occur cycle involve set resource type single instance deadlock occur thread involve cycle deadlocked case cycle graph necessary sufficient condition existence deadlock resource type instance cycle necessarily imply deadlock occur case cycle graph necessary sufficient condition existence deadlock illustrate concept return resource allocation graph depict figure 8.4 suppose thread t3 request instance resource type r2 resource instance currently available add request resource allocation graph deadlock edge t3 r2 graph figure 8.5 point minimal cycle exist system t1 r1 t2 r3 t3 r2 t1 t2 r3 t3 r2 t2 thread t1 t2 t3 deadlocked thread t2 wait resource r3 hold thread t3 thread t3 wait thread t1 thread t2 release resource r2 addition thread t1 wait thread t2 release resource r1 consider resource allocation graph figure 8.6 example cycle t1 r1 t3 r2 t1 resource allocation graph cycle deadlock deadlock observe thread t4 release instance resource type r2 resource allocate t3 break summary resource allocation graph cycle system deadlocked state cycle system deadlocked state observation important deal deadlock problem method handle deadlock generally speak deal deadlock problem ignore problem altogether pretend deadlock occur system use protocol prevent avoid deadlock ensure system enter deadlocked state allow system enter deadlocked state detect recover solution operate system include linux windows kernel application developer write program

handle deadlock typically approach outline second solution system database adopt solution allow deadlock occur manage recovery elaborate briefly method handle deadlock section 8.5 section 8.8 present detailed algorithm proceed mention researcher argue basic approach appropriate entire spectrum resource- allocation problem operate system basic approach com- bin allow select optimal approach class resource system ensure deadlock occur system use deadlock- prevention deadlock avoidance scheme deadlock prevention provide set method ensure necessary condition section 8.3.1 hold method prevent deadlock constrain request resource discuss method section 8.5 deadlock avoidance require operate system give addi- tional information advance concern resource thread request use lifetime additional knowledge operate sys- tem decide request thread wait decide current request satisfy delay system consider resource currently available resource currently allocate thread future request release thread discuss scheme section 8.6 system employ deadlock prevention deadlock-avoidance algorithm deadlock situation arise environment system provide algorithm examine state system determine deadlock occur algorithm recover deadlock deadlock occur discuss issue section 8.7 section 8.8 absence algorithm detect recover deadlock arrive situation system deadlocked state way recognize happen case undetected deadlock cause system performance deteriorate resource hold thread run thread request resource enter deadlocked state eventually system stop function need restart manually method viable approach dead- lock problem operate system mention early expense important consideration ignore possibility deadlock cheap approach system dead- lock occur infrequently month extra expense method worthwhile addition method recover liveness condition livelock recover deadlock circumstance system suffer liveness failure deadlocked state situation example real time thread run high priority thread run nonpreemptive scheduler return control operate system system manual recovery method condition simply use technique note section 8.3.1 deadlock occur neces- sary condition hold ensure condition hold prevent occurrence deadlock elaborate approach examine necessary condition separately mutual exclusion condition hold resource nonsharable sharable resource require mutually exclusive access involve

hold resource nonsharable sharable resource require mutually exclusive access involve deadlock read file good example sharable resource thread attempt open read file time grant simultaneous access file thread need wait sharable resource general prevent deadlock deny mutual exclusion condition resource intrinsically nonsharable example mutex lock simultaneously share thread hold wait ensure hold wait condition occur system guarantee thread request resource hold resource protocol use require thread request allocate resource begin execution course impractical application dynamic nature request alternative protocol allow thread request resource thread request resource use request additional resource release resource protocol main disadvantage resource utiliza- tion low resource allocate unused long period example thread allocate mutex lock entire execution require short duration second starvation possible thread need popular resource wait indefinitely resource need allocate thread necessary condition deadlock preemption resource allocate ensure condition hold use follow protocol thread hold resource request resource immediately allocate thread wait resource thread currently hold preempt word resource implicitly release preempt resource add list resource thread wait thread restart regain old resource new one request alternatively thread request resource check available allocate check allocate thread wait additional resource preempt desire resource wait thread allocate request thread resource available hold wait thread request thread wait wait resource preempt thread request thread restart allocate new resource request recover resource preempt protocol apply resource state easily save restore later cpu register database transaction can- generally apply resource mutex lock semaphore precisely type resource deadlock occur commonly option present far deadlock prevention generally impractical situation fourth final condition deadlock circular wait condition present opportunity practical solution invalidate necessary condition way ensure condition hold impose total ordering resource type require thread request resource increase order enumeration illustrate let r = r1 r2 rm set resource type assign resource type unique integer number allow compare resource determine precede ordering

type unique integer number allow compare resource determine precede ordering formally define function f r n n set natural number accomplish scheme application program develop ordering synchronization object system example lock ordering pthread program show figure 8.1 f(first mutex = 1 f(second mutex = 5

consider follow protocol prevent deadlock thread request resource increase order enumeration thread initially request instance resource ri thread request instance resource rj f(rj > f(ri example function define thread want use mutex second mutex time request mutex second mutex alternatively require thread request instance resource rj release resource ri f(ri f(rj note instance resource type need single request issue protocol circular wait condition hold demonstrate fact assume circular wait exist proof contradiction let set thread involve circular wait t0 t1 tn ti wait resource ri hold thread ti+1 modulo arithmetic index tn wait resource rn hold t0 thread ti+1 hold resource ri request resource ri+1 f(ri < f(ri+1 i. condi- tion mean f(r0 < f(r1 < < f(rn < f(r0 transitivity f(r0 < f(r0 impossible circular wait mind develop ordering hierarchy prevent deadlock application developer write program follow ordering establish lock ordering difficult especially system hundred thousand lock address challenge java developer adopt strategy method system.identityhashcode(object return default hash code value object parameter pass function order lock acquisition important note impose lock ordering guar- antee deadlock prevention lock acquire dynamically exam- ple assume function transfer fund account prevent race condition account associate mutex lock obtain lock function show figure 8.7 deadlock possible thread simultaneously invoke transaction function transpose different account thread invoke transaction(checking account saving account 25.0 invoke transaction(saving account check account 50.0 void transaction(account account

double mutex lock1 lock2 lock1 = lock(from lock2 = lock(to deadlock example lock ordering deadlock prevention algorithm discuss section 8.5 prevent dead- lock limit request limit ensure necessary condition deadlock occur possible effect prevent deadlock method low device utilization reduce system throughput alternative method avoid deadlock require additional information resource request example system resource r1 r2 system need know thread p request r1 r2 release resource thread q request r2 r1 knowledge complete sequence request release thread system decide request thread wait order avoid possible future deadlock request require make decision system consider resource currently available resource currently allocate thread future request release thread algorithm use approach differ type information require simple useful model require thread declare maximum number resource type need give priori information possible construct algorithm ensure system enter deadlocked state deadlock- avoidance algorithm dynamically examine resource allocation state ensure circular wait condition exist resource allocation state define number available allocate resource maximum demand thread follow section explore linux lockdep tool ensure resource acquire proper order responsibility kernel application developer certain software verify lock acquire proper order detect possible deadlock linux provide lockdep tool rich functionality verify locking order kernel lockdep design enable run kernel monitor usage pattern lock acquisition release set rule acquire release lock example follow note lockdep provide significantly functionality describe order lock acquire dynamically maintain system lockdep detect lock acquire order report possible deadlock condition linux spinlock interrupt handler possible source deadlock occur kernel acquire spinlock interrupt handler interrupt occur lock hold interrupt handler preempt kernel code currently hold lock spin attempt acquire lock result deadlock general strategy avoid situation disable interrupt current processor acquire spinlock interrupt handler lockdep detect interrupt enable kernel code acquire lock interrupt handler report possible deadlock scenario lockdep develop tool develop modify code kernel production system significantly slow system

tool develop modify code kernel production system significantly slow system purpose test software new device driver kernel module provide possible source deadlock designer lockdep report year development 2006 number deadlock system report reduce order

magnitude.âłž lockdep originally design use kernel recent revision tool detect deadlock user application pthreads mutex lock detail lockdep tool find state safe system allocate resource thread maximum order avoid deadlock formally system safe state exist safe sequence sequence thread < t1 t2 tn > safe sequence current allocation state ti resource request ti satisfy currently available resource plus resource hold tj j < i. situation resource ti need immediately available ti wait tj finish finish ti obtain safe unsafe deadlocked state space need resource complete designate task return allocate resource terminate ti terminate ti+1 obtain need resource sequence exist system state say unsafe safe state deadlocked state conversely deadlocked state unsafe state unsafe state deadlock figure 8.8 unsafe state lead deadlock long state safe operate system avoid unsafe deadlocked state unsafe state operate system prevent thread request resource way deadlock occur behavior thread control unsafe state illustrate consider system resource thread t0 t1 t2 thread t0 require resource thread t1 need thread t2 need resource suppose time t0 thread t0 hold resource thread t1 hold resource thread t2 hold resource free resource time t0 system safe state sequence < t1 t0 t2 >

satisfy safety condition thread t1 immediately allocate resource return system available resource thread t0 resource return system available resource finally thread t2 resource return system resource available asystem safe state unsafe state suppose time t1 thread t2 request allocate resource system long safe state point thread t1 allocate resource return system available resource thread t0 allocate resource maximum request resource wait unavailable similarly thread t2 request additional resource wait result deadlock mistake grant request thread t2 resource t2 wait thread finish release resource avoid give concept safe state define avoidance algorithm ensure system deadlock idea simply ensure system remain safe state initially system safe state thread request resource currently available system decide resource allocate immediately thread wait request grant allocation leave system scheme thread request resource currently available wait resource utilization low resource allocation system instance resource type use variant resource allocation graph define section 8.3.2 deadlock avoidance addition request assignment edge describe introduce new type edge call claim edge claim edge ti rj indicate thread ti request resource rj time future edge resemble request edge direction represent graph dashed line thread ti request resource rj claim edge ti rj convert request edge similarly resource rj release ti assignment edge rj ti reconvert claim edge ti note resource claim priori system thread ti start execute claim edge appear resource allocation graph relax condition allow claim edge ti rj add graph edge associate thread ti claim edge suppose thread ti request resource rj request grant convert request edge ti rj assignment edge rj ti result formation cycle resource allocation graph check safety cycle detection algorithm algorithm detect cycle graph require order n2 operation n number thread system cycle exist allocation resource leave system safe state cycle find allocation system unsafe state case thread ti wait request illustrate algorithm consider resource state case thread ti wait request illustrate algorithm consider resource allocation graph figure 8.9 suppose t2 request r2 r2 currently free allocate t2 action create cycle graph figure 8.10 cycle mention indicate system unsafe state t1 request r2 t2 request r1 deadlock occur resource allocation graph algorithm applicable resource- allocation system multiple instance resource type resource allocation graph deadlock avoidance deadlock avoidance algorithm describe applicable system efficient resource allocation graph scheme algorithm commonly know banker algorithm choose algorithm banking system ensure bank allocate available cash way long satisfy need customer new thread enter system declare maximum num- ber instance resource type need number exceed total number resource system user request set resource system determine allocation resource leave system safe state resource allo- cat thread wait thread release datum structure maintain implement banker algorithm datum structure encode state resource allocation system need following datum structure n number thread system m number resource type available avector length m indicate number available resource type available[j equal k k instance resource type rj unsafe state resource allocation graph max n × m matrix define maximum demand thread max[i][j equal k thread ti request k instance resource type rj allocation n × m matrix define number resource type currently allocate thread allocation[i][j equal k thread ti currently allocate k instance resource type rj need n × m matrix indicate remain resource need thread need[i][j equal k thread ti need k instance resource type rj complete task note need[i][j equal max[i][j data structure vary time size value simplify presentation banker algorithm establish notation let x y vector length n. x y x[i y[i = 1 2 n. example x = 1,7,3,2 y = 0,3,2,1 y x.

addition y < x y x y x. treat row matrix allocation need vector refer allocationi needi vector allocationi specify resource currently allocate thread ti vector needi specify additional resource thread ti request complete task present algorithm find system safe state algorithm describe follow 1 let work finish vector length m n respectively initialize work = available finish[i = false

= 0 1 n 1 2 find index a. finish[i = = false b. needi work exist step 4 3 work = work + allocationi finish[i = true step 2 4 finish[i = = true system safe state algorithm require order m × n2 operation determine state safe describe algorithm determine request safely grant let requesti request vector thread ti requesti j = = k thread ti want k instance resource type rj request resource thread ti following action take 1 requesti needi step 2 raise error condition thread exceed maximum claim 2 requesti available step 3 ti wait resource available 3 system pretend allocate request resource thread ti modify state follow available = available requesti allocationi = allocationi + requesti needi = needi requesti result resource allocation state safe transaction com- plete thread ti allocate resource new state unsafe ti wait requesti old resource allocation state restore illustrative example illustrate use banker algorithm consider system thread t0 t4 resource type b c. resource type instance resource type b instance resource type c seven instance suppose following snapshot represent current state system b c b c b c 0 1 0 7 5 3 3 3 2 2 0 0 3 2 2 3 0 2 9 0 2 2 1 1 2 2 2 0 0 2 4 3 3

content matrix need define max allocation b c 7 4 3 1 2 2 6 0 0 0 1 1 4 3 1 claim system currently safe state sequence < t1 t3 t4 t2 t0 > satisfy safety criterion suppose thread t1 request additional instance resource type instance resource type c request1 = 1,0,2 decide request immediately grant check request1 available 1,0,2 (3,3,2 true pretend request fulfil arrive following new state b c b c b c 0 1 0 7 4 3 2 3 0 3 0 2 0 2 0 3 0 2 6 0 0 2 1 1 0 1 1 0 0 2 4 3 1 determine new system state safe execute safety algorithm find sequence < t1 t3 t4 t0 t2 > satisfy safety requirement immediately grant request thread t1 able system state request 3,3,0 t4 grant resource available furthermore request 0,2,0 t0 grant resource available result state unsafe leave programming exercise student implement system employ deadlock prevention deadlock- avoidance algorithm deadlock situation occur environment system provide algorithm examine state system determine deadlock occur algorithm recover deadlock discuss requirement pertain system single instance resource type system sev- eral instance resource type point note detection recovery scheme require overhead include run time cost maintain necessary information execute detection algorithm potential loss inherent recover single instance resource type resource single instance define deadlock- detection algorithm use variant resource allocation graph call wait graph obtain graph resource allocation graph remove resource node collapse appropriate edge precisely edge ti tj wait graph imply thread ti wait thread tj release resource ti need edge ti tj resource allocation graph b correspond wait graph exist wait graph correspond resource allocation graph contain edge ti rq rq tj resource rq figure 8.11 present resource allocation graph corresponding wait deadlock exist system wait

present resource allocation graph corresponding wait deadlock exist system wait graph contain cycle detect deadlock system need maintain wait- graph periodically invoke algorithm search cycle graph algorithm detect cycle graph require o(n2 operation n number vertex graph bcc toolkit describe section 2.10.4 provide tool detect potential deadlock pthreads mutex lock user process run linux system bcc tool deadlock detector operate insert probe trace call pthread mutex lock pthread mutex unlock function specify process make function deadlock detector construct wait graph mutex lock process report possibility deadlock detect cycle graph instance resource type wait graph scheme applicable resource allocation system multiple instance resource type turn deadlock- detection algorithm applicable system algorithm employ time vary data structure similar banker algorithm section 8.6.3 available avector length m indicate number available resource type deadlock detection java thread dump java provide explicit support deadlock detection thread dump analyze run program determine deadlock thread dump useful debugging tool display snapshot state thread java application java thread dump locking information include lock block thread wait acquire thread dump generate jvm search wait graph detect cycle report deadlock detect generate thread dump run application command line enter ctrl l unix linux macos source code download text provide java example program show figure 8.1 describe generate thread dump report deadlocke java thread allocation n × m matrix define number resource type currently allocate thread request n × m matrix indicate current request thread request[i][j

equal k thread ti request k instance resource type rj relation vector define section 8.6.3 sim- plify notation treat row matrix allocation request vector refer allocationi requesti detection algo- rithm describe simply investigate possible allocation sequence thread remain complete compare algorithm banker algorithm section 8.6.3 1 let work finish vector length m n respectively initialize work = available = 0 1 n–1 allocationi 0 finish[i = false finish[i = true 2 find index a. finish[i = = false b. requesti work exist step 4 3 work = work + allocationi finish[i = true step 2 4 finish[i = = false 0 i < n system deadlocke state finish[i = = false thread ti deadlocked algorithm require order m × n2 operation detect system deadlocked state wonder reclaim resource thread ti step 3 soon determine requesti work step 2b know ti currently involve deadlock requesti work optimistic attitude assume ti require resource complete task soon return currently allocate resource system assumption incorrect deadlock occur later deadlock detect time deadlock detection algorithm illustrate algorithm consider system thread t0 t4 resource type b c. resource type seven instance resource type b instance resource type c instance follow snapshot represent current state system b c b c b c 0 1 0 0 0 0 0 0 0 2 0 0 2 0 2 3 0 3 0 0 0 2 1 1 1 0 0 0 0 2 0 0 2 claim system deadlocked state execute algorithm find sequence < t0 t2 t3 t1 t4 > result finish[i = = true i. suppose thread t2 make additional request instance type c. request matrix modify follow b c 0 0 0 2 0 2 0 0 1 1 0 0 0 0 2

claim system deadlocked reclaim resource hold thread t0 number available resource sufficient fulfill request thread deadlock exist consist thread t1 t2 t3 t4 invoke detection algorithm answer depend 1 deadlock likely occur 2 thread affect deadlock happen recovery deadlock manage deadlock database database system provide useful illustration open source commercial software manage deadlock update database perform transaction ensure data integrity lock typically transaction involve lock come surprise deadlock possible database multiple concurrent transac- tion manage deadlock transactional database system include deadlock detection recovery mechanism database server peri- odically search cycle wait graph detect deadlock set transaction deadlock detect victim select transaction abort roll release lock hold victim transaction free remain transaction deadlock remain transaction resume abort transaction reissue choice victim transaction depend database system instance mysql attempt select transaction minimize number row insert update delete deadlock occur frequently detection algorithm invoke frequently resource allocate deadlocke thread idle deadlock break addition number thread involve deadlock cycle grow deadlock occur thread make request grant immediately request final request complete chain wait thread extreme invoke deadlock-detection algorithm time request allocation grant immediately case identify deadlocke set thread specific thread cause deadlock reality deadlocke thread link cycle resource graph jointly cause deadlock different resource type request create cycle resource graph cycle complete recent request cause identifiable thread course invoke deadlock detection algorithm resource request incur considerable overhead computation time expen- sive alternative simply invoke algorithm define interval example hour cpu utilization drop 40 percent deadlock eventually cripple system throughput cause cpu utilization drop detection algorithm invoke arbitrary point time resource graph contain cycle case generally tell deadlocke thread cause deadlock recovery deadlock detection algorithm determine deadlock exist alter- native available possibility inform operator deadlock occur let operator deal deadlock manually possibility let system recover deadlock automatically option break deadlock simply abort thread break

recover deadlock automatically option break deadlock simply abort thread break circular wait preempt resource deadlocke thread process thread termination eliminate deadlock abort process thread use method method system reclaim resource allocate abort deadlocke process method clearly break dead- lock cycle great expense deadlocke process com- put long time result partial computation discard probably recompute later abort process time deadlock cycle eliminate method incur considerable overhead process abort deadlock detection algorithm invoke determine process deadlocked abort process easy process midst update file terminate leave file incorrect state similarly process midst update share datum hold mutex lock system restore status lock available guarantee integrity share datum partial termination method determine deadlocke process process terminate determination policy decision similar cpu scheduling decision question basically economic abort process termination incur minimum cost unfortunately term minimum cost precise factor affect process choose include 1 priority process 2 long process compute long process compute complete designate task 3 type resource process exam- ple resource simple preempt 4 resource process need order complete 5 process need terminate eliminate deadlock resource preemption successively preempt resource process resource process deadlock cycle break preemption require deal deadlock issue need 1 select victim resource process pre- empte process termination determine order pre- emption minimize cost cost factor include parameter number resource deadlocke process hold time process far consume 2 rollback preempt resource process process clearly continue normal execution miss need resource roll process safe state restart state general difficult determine safe state simple solution total rollback abort process restart effective roll process far necessary break deadlock method require system information state run process 3 starvation ensure starvation occur guarantee resource preempt system victim selection base primarily cost factor happen process pick victim result process complete designate task starvation situation practical system address clearly ensure process pick victim small finite number time

address clearly ensure process pick victim small finite number time common solution include number rollback deadlock occur set process process set wait event cause process set necessary condition deadlock 1 mutual exclusion 2 hold wait 3 preemption 4 circular wait deadlock possible condition present deadlock model resource allocation graph cycle deadlock prevent ensure necessary condition deadlock occur necessary condition eliminate circular wait practical approach deadlock avoid banker algorithm deadlock possible deadlock detection algorithm evaluate process resource run system determine set process deadlocked state deadlock occur system attempt recover deadlock abort process circular wait preempt resource assign deadlocke process list example deadlock relate computer- suppose system unsafe state possible thread complete execution enter deadlocked consider following snapshot system b c d b c d b c d 0 0 1 2 0 0 1 2 1 5 2 0 1 0 0 0 1 7 5 0 1 3 5 4 2 3 5 6 0 6 3 2 0 6 5 2 0 0 1 4 0 6 5 6

answer follow question banker algorithm content matrix need system safe state request thread t1 arrive 0,4,2,0 request possible method prevent deadlock single high order resource request resource example multiple thread attempt access synchronization object e deadlock possible synchronization object include mutexe semaphore condition variable like prevent deadlock add sixth object f. thread want acquire synchronization lock object e acquire lock object f. solution know containment lock object e contain lock object f.

compare scheme circular wait scheme section 8.5.4 prove safety algorithm present section 8.6.3 require order m × n2 operation consider computer system run 5,000 job month deadlock prevention deadlock avoidance scheme deadlock occur twice month operator terminate rerun job deadlock job worth dollar cpu time job terminate tend half system programmer estimate deadlock avoidance algorithm like banker algorithm instal system increase 10 percent average execution time job machine currently 30 percent idle time 5,000 job month run turnaround time increase 20 percent average argument instal deadlock avoidance argument instal deadlock avoidance system detect thread starve answer yes explain answer explain system deal starvation problem consider following resource allocation policy request release resource allow time request resource satisfy resource available check thread block wait resource block thread desire resource resource take away give request thread vector resource block thread wait increase include resource take away example system resource type vector avail- able initialize 4,2,2 thread t0 ask 2,2,1 get t1 ask 1,0,1 get t0 ask 0,0,1 block resource available t2 ask 2,0,0 get available 1,0,0 allocate t0 t0 block t0 allocation vector go 1,2,1 need vector go deadlock occur answer yes example answer specify necessary condition occur indefinite blocking occur explain answer consider following snapshot system b c d b c d 3 0 1 4 5 1 1 7 2 2 1 0 3 2 1 1 3 1 2 1 3 3 2 1 0 5 1 0 4 6 1 2 4 2 1 2 6 3 2 5 banker algorithm determine follow state unsafe state safe illustrate order thread complete illustrate state unsafe available = 0 3 0 1 available = 1 0 0 2 suppose code deadlock avoidance safety algorithm determine system safe state ask implement deadlock detection algorithm simply safety algorithm code redefine maxi =

deadlock detection algorithm simply safety algorithm code redefine maxi = waitingi + allocationi waitingi vector specify resource thread wait allocationi define section 8.6 explain answer possible deadlock involve single threaded process explain answer research involve deadlock conduct year ago dijkstra 1965 influential contributor deadlock detail mysql database manage deadlock find detail lockdep tool find https://www.kernel.org/doc/ e. w. dijkstra

cooperate sequential processes technical report technological university eindhoven netherlands 1965 chapter 8 exercise consider traffic deadlock depict figure 8.12 necessary condition deadlock hold state simple rule avoid deadlock system draw resource allocation graph illustrate deadlock program example show figure 8.1 section 8.2 section 6.8.1 describe potential deadlock scenario involv- e process p0 p1 semaphore s q. draw resource- allocation graph illustrate deadlock scenario present section assume multithreaded application use reader writer lock synchronization apply necessary condition dead- lock deadlock possible multiple reader writer lock program example show figure 8.1 lead dead- lock

describe role cpu scheduler play con- tribute deadlock program section 8.5.4 describe situation prevent deadlock ensure lock acquire certain order point deadlock possible situation thread simultaneously invoke transaction function fix transac- tion function prevent deadlock traffic deadlock exercise 8.12 resource allocation graph show figure 8.12 illus- trate deadlock situation deadlocke provide cycle thread resource deadlock situation illustrate order thread complete execution compare circular wait scheme deadlock avoidance scheme like banker algorithm respect follow real computer system resource available demand thread resource consistent long period month resource break replace new process thread come new resource buy add system deadlock control banker algorithm resource allocation graph exercise 8.18 follow change safely introduce possibility deadlock circumstance increase available new resource add decrease available resource permanently remove system increase max thread thread need want resource allow decrease max thread thread decide need resource increase number thread decrease number thread consider follow snapshot system b c d b c d 2 1 0 6 6 3 2 7 3 3 1 3 5 4 1 5 2 3 1 2 6 6 1 4 1 2 3 4 4 3 4 5 3 0 3 0 7 2 6 1 content need matrix consider system consist resource type share thread need resource system deadlock free consider system consist m resource type share n thread thread request release resource time system deadlock free follow maximum need thread resource m sum maximum need m + n.

consider version dining philosopher problem chopstick place center table philosopher assume request chopstick time describe simple rule determine particular request satisfy cause deadlock give current allocation chopstick philosopher consider setting precede exercise assume philosopher require chopstick eat resource request issue time describe simple rule determine particular request satisfy cause deadlock give current allocation chopstick philosopher obtain banker algorithm single resource type general banker algorithm simply reduce dimensionality array 1 example implement multiple- resource type banker scheme apply single resource type scheme resource type individually consider follow snapshot system b c d b c d 1 2 0 2 4 3 1 6 0 1 1 2 2 4 2 4 1 2 4 0 3 6 5 1 1 2 0 1 2 6 2 3 1 0 0 1 3 1 1 2 banker algorithm determine follow state unsafe state safe illustrate order thread complete illustrate state unsafe available = 2 2 2 3 available = 4 4 1 1 available = 3 0 1 4 available = 1 5 2 2 consider following snapshot system b c d b c d b c d 3 1 4 1 6 4 7 3 2 2 2 4 2 1 0 2 4 2 3 2 2 4 1 3 2 5 3 3 4

1 1 0 6 3 3 2 2 2 2 1 5 6 7 5 answer following question banker algorithm illustrate system safe state demonstrate order thread complete request thread t4 arrive 2 2 2 4 request request thread t2 arrive 0 1 1 0 request request thread t3 arrive 2 2 1 2

request optimistic assumption deadlock detection algo- rithm assumption violate single lane bridge connect vermont village north tun- bridge south tunbridge farmer village use bridge deliver produce neighbor town bridge deadlocked northbound southbound farmer bridge time vermont farmer stubborn unable semaphore and/or mutex lock design algorithm pseudocode prevent deadlock initially concern starvation situation northbound farmer prevent southbound farmer bridge vice versa modify solution exercise 8.30 starvation free implement solution exercise 8.30 posix synchronization particular represent northbound southbound farmer separate thread farmer bridge associate thread sleep random period time represent travel bridge design program create thread represent northbound southbound farmer figure 8.7 illustrate transaction function dynamically acquire lock text describe function present difficulty acquire lock way avoid deadlock java implementation transaction provide source code download text modify sys- tem.identityhashcode method lock acquire project write program implement banker algo- rithm discuss section 8.6.3 customer request release resource bank banker grant request leave system safe state request leave system unsafe state deny code example describe project illustrate c develop solution java banker consider request n customer m resource type outline section 8.6.3 banker track resource following data structure define number customer 5 define number resource 4 available resource int available[number resource /*the maximum demand customer int maximum[number customers][number resource currently allocate customer int allocation[number customers][number resource remaining need customer int need[number customers][number resource banker grant request satisfy safety algorithm outline section 8.6.3.1 request leave system safe state banker deny function prototype request release resource int request resources(int customer num int request void release resources(int customer num int release request resource function return 0 successful 1

test implementation design program allow user interactively enter request resource release resource output value different data structure available maximum allocation need invoke program pass number resource type command line example resource type instance type second type seven type fourth type invoke program follow ./a.out 10 5 7 8 available array initialize value program initially read file contain maximum number request customer example customer resource input file appear follow line input file represent maximum request resource type customer program initialize maximum array value program user enter command respond request resource release resource current value different datum structure use command rq request resource rl releas- e resource output value different data structure example customer 0 request resource 3 1 2 1 following command enter rq 0 3 1 2 1 program output request satisfy deny safety algorithm outline section 8.6.3.1 similarly customer 4 release resource 1 2 3 1 user enter following command rl 4 1 2 3 1

finally command enter program output value available maximum allocation need array main purpose computer system execute program program datum access partially main memory execution modern computer system maintain process memory system execution memory management scheme exist reflecte approach effectiveness algorithm vary situation selection memory management scheme system depend factor especially system hardware design algorithm require form hardware support c h p t e r chapter 5 show cpu share set process result cpu scheduling improve utilization cpu speed computer response user realize increase performance process memory share memory chapter discuss way manage memory memory-management algorithm vary primitive bare machine approach strategy use paging approach advantage disad- vantage selection memory management method specific system depend factor especially hardware design system shall algorithm require hardware support lead system closely integrate hardware operating system memory explain difference logical physical address role memory management unit mmu translate address apply first- best- bad fit strategy allocate memory contigu- explain distinction internal external fragmentation translate logical physical address paging system include translation look aside buffer tlb describe hierarchical paging hashed paging inverted page table describe address translation ia-32 x86 64 armv8 architecture see chapter 1 memory central operation modern computer system memory consist large array byte address cpu fetch instruction memory accord value program counter instruction cause additional loading store specific memory address typical instruction execution cycle example fetch instruc- tion memory instruction decode cause operand fetch memory instruction execute operand result store memory memory unit see stream memory address know generate instruction counter indexing indirection literal address instruction datum accordingly ignore program generate memory address interested sequence memory address generate run program begin discussion cover issue pertinent manage memory basic hardware binding symbolic virtual mem- ory address actual physical address distinction logical physical address conclude section discussion dynamic linking share library main memory register

conclude section discussion dynamic linking share library main memory register build processing core general purpose storage cpu access directly machine instruction memory address argument disk address instruction execution datum instruction direct access storage device datum memory move cpu operate register build cpu core generally accessible cycle cpu clock cpu core decode instruction per- form simple operation register content rate opera- tion clock tick say main memory access transaction memory bus complete memory access cycle cpu clock case processor normally need stall datum require complete instruction execute situation intolerable frequency memory access remedy add fast memory cpu main mem- ory typically cpu chip fast access cache describe section 1.5.5 manage cache build cpu hardware automatically speed memory access operate system control recall section 5.5.2 memory stall multithreaded core switch stall hardware thread hardware thread concern relative speed access physi- cal memory ensure correct operation proper system operation protect operate system access user pro- cesse protect user process protection provide hardware operate system usually intervene cpu memory access result performance penalty hardware implement production differ- ent way chapter outline possible base + limit base limit register define logical address space need sure process separate memory space separate process memory space protect process fundamental have multiple process load memory concurrent execution separate memory space need ability determine range legal address process access ensure process access legal address provide protection register usually base limit illustrate figure 9.1 base register hold small legal physical memory address limit register specify size range example base register hold 300040 limit register 120900 program legally access address 300040 420939 inclusive protection memory space accomplish have cpu hardware compare address generate user mode register attempt program execute user mode access operating system memory user memory result trap operate system treat attempt fatal error figure 9.2 scheme prevent user program accidentally

attempt fatal error figure 9.2 scheme prevent user program accidentally deliberately modify code datum structure operate system user base limit register load operate system use special privileged instruction privileged instruction execute kernel mode operate system execute kernel mode operate system load base limit register scheme allow operate system change value register prevent user program change register content operate system execute kernel mode give unrestricted access operate system memory user memory provision allow operate system load user program user memory dump program case error access modify parameter system call perform o user memory provide service consider example operate system trap operate system illegal address error base + limit hardware address protection base limit register multiprocessing system execute context switch store state process register main memory load process context main memory register usually program reside disk binary executable file run program bring memory place context process describe section 2.5 eligible execution available cpu process execute access instruction datum memory eventually process terminate memory reclaim use process system allow user process reside physical memory address space computer start 00000

address user process need 00000 later operate system actually place process physical memory case user program go step optional execute figure 9.3 address repre- sente different way step address source program generally symbolic variable count acompiler typically bind symbolic address relocatable address 14 byte beginning module linker loader section 2.5 turn bind relocatable address absolute address 74014 binding mapping address space classically binding instruction datum memory address step way compile time

know compile time process reside memory absolute code generate example know user process reside start location r generate compiler code start location extend later time start location change necessary recompile code multistep processing user program load time know compile time process reside memory compiler generate relocatable code case final binding delay load time start address change need reload user code incorporate change value execution time process move execution memory segment binding delay run time special hardware available scheme work discuss section 9.1.3 operate system use method major portion chapter devote show bind- ing implement effectively computer system discuss appropriate hardware support logical versus physical address space address generate cpu commonly refer logical address address see memory unit load memory management unit mmu memory address register memory commonly refer bind address compile load time generate identical logical physical address execution time address binding scheme result differ logical physical address case usually refer logical address virtual address use logical address virtual address interchangeably text set logical address generate program logical address space set physical address correspond logical address physical address space execution time address bind scheme logical physical address run time mapping virtual physical address hardware device call memory management unit mmu figure 9.4 choose different method accomplish mapping discuss section 9.2 section 9.3 time illustrate mapping simple mmu scheme generalization base- register scheme describe section 9.1.1 base register call relocation register value relocation register add address generate user process time address send memory figure 9.5 example base 14000 attempt user address location 0 dynamically relocate location 14000 access location 346 map location 14346 user program access real physical address program create pointer location 346

store memory manipulate com- pare address number 346 memory address indirect load store relocate relative base register user program deal logical address memory- map hardware convert logical address physical address form execution time binding discuss section 9.1.2 final loca- tion reference memory address determine reference different type address logical address range 0 max physical address range r + 0 r + max base value r user program generate logical address think process run memory location 0

max logical address map physical address dynamic relocation relocation register concept logical address space bind separate physical address space central proper memory management discussion far necessary entire program datum process physical memory process execute size process limit size physical memory obtain well memory space utilization use dynamic loading dynamic loading routine load call routine keep disk relocatable load format main program load memory execute routine need routine call routine check routine load relocatable link loader call load desire routine memory update program address table reflect change control pass newly load routine advantage dynamic loading routine load need method particularly useful large amount code need handle infrequently occur case error routine situation total program size large portion load small dynamic loading require special support operate system responsibility user design program advantage method operate system help programmer provide library routine implement dynamic loading dynamic linking shared libraries dynamically link library dll system library link user program program run refer figure 9.3 operate system support static linking system library treat like object module combine loader binary program image dynamic linking contrast similar dynamic loading linking loading postpone execution time feature usually system library standard c language library facility program system include copy language library routine reference program executable image requirement increase size executable image waste main memory second advantage dll library share multiple process instance dll main memory reason dll know share library extensively windows linux program reference routine dynamic library loader locate dll load memory necessary adjust address reference function dynamic library location memory dll store dynamically link library extend library update bug fix addition library replace new version program reference library automatically use new version dynamic linking program need relinke gain access

new library program accidentally execute new incompatible version library version information include program library version library load memory program use version information decide copy library use version minor change retain version number version major change increment number program compile new library version affect incompatible change incorporate program link new library instal continue old library unlike dynamic loading dynamic linking share library generally require help operate system process memory pro- tecte operate system entity check need routine process memory space allow multiple process access memory address elaborate concept dll share multiple process discuss paging section 9.3.4 contiguous memory allocation main memory accommodate operate system user process need allocate main memory efficient way possible section explain early method contiguous memory usually divide partition operate system user process place operate system low memory address high memory address decision depend factor location interrupt vector operate system include linux windows place operate system high memory discuss situation contiguous memory allocation usually want user process reside memory time need consider allocate available memory process wait bring memory contiguous mem- ory allocation process contain single section memory contiguous section contain process discuss memory allocation scheme address issue prevent process access memory combine idea previously discuss system relo- cation register section 9.1.3 limit register section 9.1.1 accomplish goal relocation register contain value small physical address limit register contain range logical address example relocation = 100040 limit = 74600 logical address fall range specify limit register mmu map log- ical address dynamically add value relocation register map address send memory figure 9.6 cpu scheduler select process execution dispatcher load relocation limit register correct value context switch address generate cpu check register protect operate system user program datum modify run process relocation register scheme provide effective way allow oper- ating system size change dynamically flexibility desirable situation example operate system

system size change dynamically flexibility desirable situation example operate system contain code buffer space device driver device driver currently use make little sense memory instead load memory need likewise device driver long need remove memory allocate need trap address error hardware support relocation limit register ready turn memory allocation simple method allocate memory assign process variably sized partition mem- ory partition contain exactly process variable- partition scheme operate system keep table indicate part memory available occupy initially memory available user process consider large block available memory hole eventually memory contain set hole figure 9.7 depict scheme initially memory fully utilize con- taine process 5 8 2 process 8 leave contiguous hole later process 9 arrive allocate memory process 5 depart result noncontiguous hole process enter system operate system take account memory requirement process available memory space determine process allocate memory process allocate space load memory compete cpu time process terminate release memory operate system provide process happen sufficient memory satisfy demand arrive process option simply reject process provide appropriate error message alternatively place process wait queue memory later release operate system check wait queue determine satisfy memory demand wait general mention memory block available comprise set hole size scatter memory process arrive need memory system search set hole large process hole large split part allocate arrive process return set hole process terminate release block memory place set hole new hole adjacent hole adjacent hole merge form large hole procedure particular instance general dynamic storage- allocation problem concern satisfy request size n list free hole solution problem fit good fi bad fi strategy one commonly select free hole set available hole contiguous memory allocation fit allocate hole big search start beginning set hole location previous fit search end stop search soon find free hole large good fi allocate small hole big

find free hole large good fi allocate small hole big search entire list list order size strategy produce small leftover hole bad fit allocate large hole search entire list sort size strategy produce large leftover hole useful small leftover hole good fit simulation show fit good fit well bad fit term decrease time storage utilization fit good fit clearly well term storage utilization fit fit good fit strategy memory allocation suffer exter- nal fragmentation process load remove memory free memory space break little piece external fragmentation exist total memory space satisfy request available space contiguous storage fragment large number small hole fragmentation problem severe bad case block free wasted memory process small piece memory big free block instead able run process fit good fit strategy affect fragmentation fit well system good fit well factor end free block allocate leftover piece matter algorithm external fragmentation problem depend total memory storage average process size external fragmentation minor major problem statistical analysis fit instance reveal optimization give n allocate block 0.5 n block lose fragmentation memory unusable property know memory fragmentation internal external consider multiple partition allocation scheme hole 18,464 byte suppose process request 18,462 byte allocate exactly request block leave hole 2 byte overhead track hole substantially large hole general approach avoid problem break physical memory fix sized block allocate memory unit base block size approach memory allocate process slightly large request memory difference number internal fragmentation unused memory internal partition solution problem external fragmentation compaction goal shuffle memory content place free memory large block compaction possible relocation static assembly load time compaction possible relocation dynamic execution time address relocate dynamically relocation require move program datum change base register reflect new base address compaction possible determine cost simple compaction algorithm process end memory hole direction produce large hole available memory scheme expensive

memory hole direction produce large hole available memory scheme expensive possible solution external fragmentation problem per- mit logical address space process noncontiguous allow process allocate physical memory memory available strategy paging common memory management technique computer system describe paging follow section fragmentation general problem computing occur manage block datum discuss topic storage management chapter chapter 11 chapter 15 memory management discuss far require physical address space process contiguous introduce paging memory- management scheme permit process physical address space non- contiguous paging avoid external fragmentation associate need compaction problem plague contiguous memory allocation offer numerous advantage

paging form oper- ating system large server mobile device paging implement cooperation operate system computer hardware basic method implement paging involve break physical mem- ory fixed sized block call frame break logical memory block size call page process execute page load available memory frame source file system back store back store divide fix sized block size memory frame cluster multiple frame simple idea great functionality wide ramification example logical address space totally separate physical address space process logical 64 bit address space system 264 byte physical memory address generate cpu divide part page number p page offset d page number index process page table illustrate figure 9.8 page table contain base address frame physical memory offset location frame reference base address frame combine page offset define physical memory address paging model memory show figure following outline step take mmu translate logical address generate cpu physical address 1 extract page number p use index page table 2 extract corresponding frame number f page table 3 replace page number p logical address frame number offset d change replace frame number offset comprise physical address page size like frame size define hardware size page power 2 typically vary 4 kb 1 gb page depend computer architecture selection power 2 page size make translation logical address page number page offset particularly easy size logical address space 2 m page size 2n byte high order mn bit logical address designate page number n low order bit designate page offset logical address follow m n

paging model logical physical memory p index page table d displacement concrete minuscule example consider memory figure 9.10 logical address n = 2 m = 4 page size 4 byte physical memory 32 byte 8 page programmer view memory map physical memory logical address 0 page 0 offset 0 index page table find page 0 frame 5 logical address 0 map physical address 20 [= 5 × 4 + 0 logical address 3 page 0 offset 3 map physical address 23 [= 5 × 4 + 3 logical address 4 page 1 offset 0 accord page table page 1 map frame 6 logical address 4 map physical address 24 [= 6 × 4 + 0 logical address 13 map physical address 9 notice paging form dynamic relocation logical address bind paging hardware physical address paging similar table base relocation register frame memory use paging scheme external fragmentation free frame allocate process need internal fragmentation notice frame allocate unit memory requirement process happen coincide page bound- arie frame allocate completely example page size 2,048 byte process 72,766 byte need 35 page plus 1,086 byte allocate 36 frame result internal fragmentation 2,048 1,086 = 962 byte bad case process need n page plus 1 byte allocate n + 1 frame result internal fragmentation entire frame paging example 32 byte memory 4 byte page process size independent page size expect internal fragmen- tation average half page process consideration suggest small page size desirable overhead involve page- table entry overhead reduce size page increase disk o efficient datum transfer large chapter 11 generally page size grow time process datum set main memory large today page typically 4 kb 8 kb size system support large page size cpu operating system support multiple page size instance x86 64 system windows 10 support page size 4 kb 2 mb linux support page size default page size typically 4 kb architecture- dependent large page size call huge page frequently 32 bit cpu page table entry 4 byte long size vary 32 bit entry point 232 physical page frame frame size 4 kb 212 system 4 byte entry address 244 byte 16 tb physical memory note size physical memory page memory system typically different maximum logical size process explore paging obtain page size linux system linux system page size vary accord architecture way obtain page size approach use system getpagesize strategy enter following command command line technique return page size number byte introduce information keep page table entry information reduce number bit available address page frame system 32 bit page table entry address physical memory possible maximum process arrive system execute size express page examine page process need frame process require n page n frame available memory n frame available allocate arrive process page process load allocate frame frame number page table process page load frame frame number page table figure 9.11 important aspect paging clear separation pro- grammer view memory actual physical memory programmer view memory single space contain program fact user program scatter physical memory hold 13 page 1 new process page table free frame allocation b allocation program difference programmer view memory actual physical memory reconcile address translation hard- ware logical address translate physical address map-

address translation hard- ware logical address translate physical address map- ping hide programmer control operate system notice user process definition unable access memory way address memory outside page table table include page process own operate system manage physical memory aware allocation detail physical memory frame allocate frame available total frame information generally keep single system wide data structure call frame table frame table entry physical page frame indicate free allocate allocate page process process addition operate system aware user process oper- eat user space logical address map produce physical address user make system o example provide address parameter buffer instance address map produce correct physical address operate system maintain copy page table process maintain copy instruction counter register content copy translate logical address physical address operate system map logical address physical address manually cpu dispatcher define hardware page table process allocate cpu paging increase context switch time page table process datum structure pointer page table store register value like instruction pointer process control block process cpu scheduler select process execution reload user register appropriate hardware page table value store user page table hardware implementation page table way simple case page table implement set dedicated high speed hardware register make page address translation efficient approach increase context switch time register exchange context switch use register page table satisfactory page table rea- sonably small example 256 entry contemporary cpu support large page table example 220 entry machine use fast register implement page table feasible page table keep main memory page table base register ptbr point page table change page table require change register substantially reduce context switch time translation look aside buffer store page table main memory yield fast context switch result slow memory access time suppose want access location i. index page table value ptbr offset page number i. task require

page table value ptbr offset page number i. task require memory access provide frame number combine page offset produce actual address access desire place memory scheme memory access need access datum page table entry actual datum memory access slow factor 2 delay consider intolerable circumstance standard solution problem use special small fast lookup hardware cache call translation look aside buffer tlb tlb asso- ciative high speed memory entry tlb consist part key tag value associative memory present item item compare key simultaneously item find cor- respond value field return search fast tlb lookup modern hardware instruction pipeline essentially add performance penalty able execute search pipeline step tlb keep small typically 32 1,024 entry size cpu implement separate instruction datum address tlb double number tlb entry available lookup occur different pipeline step development example evolution cpu technology system evolve have tlb have multiple level tlb multiple level cache tlb page table following way tlb contain page table entry logical address generate cpu mmu check page number present tlb page number find frame number immediately available access memory mention step execute instruction pipeline cpu add performance penalty compare system implement paging page number tlb know tlb miss address translation proceed follow step illustrate section 9.3.1 memory reference page table frame number obtain use access memory figure 9.12 addition add page number frame number tlb find quickly reference tlb entry exist entry select replacement replacement policy range recently lru round robin random cpu allow operate system par- ticipate lru entry replacement handle matter furthermore tlbs allow certain entry wire mean remove tlb typically tlb entry key kernel code wire tlbs store address space identifier asid tlb entry asid uniquely identify process provide address space protection process tlb attempt resolve virtual page num- ber ensure asid currently run process match asid associate virtual page

ensure asid currently run process match asid associate virtual page asid match attempt treat tlb miss addition provide address space protection asid allow tlb contain entry different process simulta- neously tlb support separate asid time new page table select instance context switch tlb flushe erase ensure execute process use paging hardware tlb wrong translation information tlb include old entry contain valid virtual address incorrect invalid physical address leave previous process percentage time page number interest find tlb call hit ratio 80 percent hit ratio example mean find desire page number tlb 80 percent time take 10 nanosecond access memory map memory access take 10 nanosecond page number tlb fail find page number tlb access memory page table frame number 10 nanosecond access desire byte memory 10 nanosecond total 20 nanosecond assume page- table lookup take memory access shall find effective memory access time weight case effective access time = 0.80 × 10 + 0.20 × 20 = 12 nanosecond example suffer 20 percent slowdown average memory access time 10 12 nanosecond 99 percent hit ratio realistic effective access time = 0.99 × 10 + 0.01 × 20 = 10.1 nanosecond increase hit rate produce 1 percent slowdown access time

note early cpu today provide multiple level tlb calculat- e memory access time modern cpu complicated show example instance intel core i7 cpu 128 entry l1 instruction tlb 64 entry l1 data tlb case miss l1 take cpu cycle check entry l2 512 entry tlb miss l2 mean cpu walk page table entry memory find associate frame address hundred cycle interrupt operate system work complete performance analysis paging overhead system require miss rate information tlb tier general information hardware feature signif- icant effect memory performance operating system improvement paging result turn affect hardware change tlbs explore impact hit ratio tlb tlbs hardware feature little concern operate system designer designer need understand function feature tlbs vary hardware platform opti- mal operation operating system design give platform imple- ment paging accord platform tlb design likewise change tlb design example different generation intel cpu necessitate change paging implementation operate system use memory protection page environment accomplish protection bit associate frame normally bit keep page table bit define page read write read reference memory go page table find correct frame number time physical address compute protection bit check verify write read page attempt write read page cause hardware trap operate system memory protection violation easily expand approach provide fine level protection create hardware provide read read write execute protection provide separate protection bit kind access allow combination access illegal attempt trap operate system additional bit generally attach entry page table valid invalid bit bit set valid associated page process logical address space legal valid page bit set invalid page process logical address space illegal address trap use valid invalid bit operate system set bit page allow disallow access page suppose example system 14 bit address space 0 16383 program use address 0 10468

give page size 2 kb situation show figure 9.13 address page 0 1 2 3 4 5

map normally page table attempt generate address page 6 7 find valid v invalid bit page table valid invalid bit set invalid computer trap operate system invalid page reference notice scheme create problem program extend address 10468 reference address illegal reference page 5 classify valid access address 12287 valid address 12288 16383 invalid problem result 2 kb page size reflect internal fragmentation rarely process use address range fact process use small fraction address space available wasteful case create page table entry page address range table unused valuable memory space system provide hardware form page table length register ptlr indicate size page table value check logical address verify address valid range process failure test cause error trap advantage paging possibility share common code considera- tion particularly important environment multiple process consider standard c library provide portion system interface version unix linux typical linux system user process require standard c library libc option process load copy libc address space system 40 user process libc library 2 mb require 80 mb code reentrant code share show figure 9.14 process share page standard c library libc figure show libc library occupy page reality occupy reentrant code non self modifying code change execution process execute code time process copy register datum storage hold datum process execution datum different process course different copy standard c library need keep physical memory page table user process map physical copy libc support 40 process need copy library total space require 2 mb instead 80 mb significant saving addition run time library libc heavily program share compiler window system database system share library discuss section 9.1.5 typically implement share page sharable code reentrant read nature share code leave correctness code operate system enforce property p 1 process p 1 p 2 process p 2 p 3 process p 3 sharing

2 process p 2 p 3 process p 3 sharing standard c library paging environment structure page table sharing memory process system similar sharing address space task thread describe chapter 4 furthermore recall chapter 3 describe share memory method interprocess communication operate system implement share memory share page organize memory accord page provide numerous benefit addition allow process share physical page cover benefit chapter 10 structure page table section explore common technique structure page table include hierarchical paging hash page table inverted modern computer system support large logical address space 232 264 environment page table excessively large example consider system 32 bit logical address space page size system 4 kb 212 page table consist 1 million entry 220 = 232/212 assume entry consist 4 byte process need 4 mb physical address space page table clearly want allocate page table contiguously main memory simple solution problem divide page table small piece accomplish division way way use level paging algorithm page table page figure 9.15 example consider system 32 bit logical address space page size 4 kb logical address divide page number consist 20 bit page offset consist 12 bit page page table page number divide 10 bit page number 10 bit page offset logical address p1 index outer page table p2 displacement page inner page table address translation method architecture show figure 9.16 address translation work outer page table inward scheme know forward mapped system 64 bit logical address space level paging scheme long appropriate illustrate point let suppose page size system 4 kb 212 case page table consist 252 entry use level paging scheme inner page table conveniently page long contain 210 4 byte entry address look like level page table scheme outer page table consist 242 entry 244 byte obvious way avoid large table divide outer page table small

obvious way avoid large table divide outer page table small piece approach 32 bit processor add flexibility divide outer page table way example page outer page table give level paging scheme suppose outer page table standard size page 210 entry 212 byte case 64 bit address space daunting 2nd outer page outer page table 234 byte 16 gb size step level paging scheme second level outer page table page forth 64 bit ultrasparc require seven level paging prohibitive number memory access structure page table address translation level 32 bit paging architecture translate logical address example 64 bit architecture hierarchical page table generally consider inappropriate hash page table approach handle address space large 32 bit use hash page table hash value virtual page number entry hash table contain link list element hash location handle collision element consist field 1 virtual page number 2 value map page frame 3 pointer element link list algorithm work follow virtual page number virtual address hash hash table virtual page number compare field 1 element link list match correspond page frame field 2 form desire physical address match subsequent entry link list search match virtual page number scheme show figure 9.17 hash page table variation scheme useful 64 bit address space propose variation use cluster page table similar hash page table entry hash table refer page 16 single page single page table entry store mapping multiple physical page frame clustered page table particularly useful sparse address space memory reference noncontiguous scatter address space inverted page table usually process associate page table page table entry page process slot virtual address regardless validity table representation natural process reference page page virtual address operate system translate reference physical memory address table sort virtual address operate system able calculate table associated physical address entry locate use value directly drawback method

associated physical address entry locate use value directly drawback method page table consist million entry table consume large amount physical memory track physical memory solve problem use inverted page table inverted page table entry real page frame memory entry consist virtual address page store real memory location information process own page page table system entry page physical memory figure 9.18 show operation inverted page table compare figure 9.8

depict standard page table operation inverted page table require address space identifier section 9.3.2 store entry page table table usually contain invert page table structure page table different address space map physical memory store address space identifier ensure logical page particular process map corresponding physical page frame example system inverted page table include 64 bit ultrasparc powerpc illustrate method describe simplified version inverted system consist triple < process id page number offset > inverted page table entry pair < process id page number > process id assume role address space identifier memory reference occur virtual address consist < process id page- number > present memory subsystem inverted page table search match match find entry physical address < offset > generate match find illegal address access attempt scheme decrease memory need store page table increase time need search table page reference occur inverted page table sort physical address lookup occur virtual address table need search match find search far long alleviate problem use hash table describe section 9.4.2 limit search page table entry course access hash table

add memory reference procedure virtual memory reference require real memory read hash table entry page table recall tlb search hash table consult offer performance improvement interesting issue inverted page table involve share memory standard paging process page table allow multiple virtual address map physical address method inverted page table virtual page entry physical page physical page share virtual address inverted page table mapping virtual address share physical address occur give time reference process share memory result page fault replace mapping different virtual address oracle sparc solaris consider final example modern 64 bit cpu operate system tightly integrate provide low overhead virtual memory solaris run sparc cpu fully 64 bit operate system solve problem virtual memory physical memory keep multiple level page table approach bit complex solve problem efficiently hashed page table hash table kernel user process map memory address virtual physical memory hash table entry represent contiguous area map virtual memory efficient have separate hash table entry page entry base address span indicate number page entry represent virtual physical translation long address require search hash table cpu implement tlb hold transla- tion table entry ttes fast hardware lookup acache tte reside translation storage buffer tsb include entry recently access page virtual address reference occur hardware search tlb translation find hardware walk in- memory tsb look tte correspond virtual address cause lookup tlb walk functionality find modern cpu match find tsb cpu copy tsb entry tlb memory translation complete match find tsb kernel interrupt search hash table kernel create tte appropriate hash table store tsb automatic loading tlb cpu memory management unit finally interrupt han- dler return control mmu complete address translation retrieve request byte word main memory process instruction datum operate memory execute process portion process swap temporarily memory back store bring memory continue execution figure 9.19 swapping make possible

store bring memory continue execution figure 9.19 swapping make possible total physical address space process exceed real physical memory system increase degree multiprogramming standard swapping process disk back store standard swapping involve move entire process main memory back store back store commonly fast secondary storage large accommodate part process need store retrieve provide direct access memory image process swap back store data structure associate process write back store multithreaded process thread data structure swap operate system maintain metadata process swap restore swap memory advantage standard swapping allow physical memory oversubscribe system accommodate process actual physical memory store idle idle process good candidate swapping memory allocate inactive process dedicate active process inactive process swap active swap illustrate figure 9.19 swapping paging standard swapping traditional unix system generally long contemporary operate system time require entire process memory backing store prohibitive exception solaris use standard swapping dire circumstance available memory extremely system include linux windows use variation swap- ping page process entire process swap strategy allow physical memory oversubscribe incur cost swap entire process presumably small number page involve swapping fact term swapping generally refer standard swapping paging refer swapping paging page operation move page memory backing store reverse process know page swap paging illus- trate figure 9.20 subset page process b page page respectively shall chapter 10 swap paging work conjunction virtual memory swap mobile systems operate system pc server support swap page con- trast mobile system typically support swap form mobile device generally use flash memory spacious hard disk nonvolatile storage result space constraint reason mobile operating system designer avoid swap reason include limited number write flash memory tolerate unreliable poor throughput main memory flash memory swap paging instead swapping free memory fall certain thresh- old apple ios ask application voluntarily relinquish allocate mem- ory read datum code remove main memory

relinquish allocate mem- ory read datum code remove main memory later reload flash memory necessary datum modify stack remove application fail free sufficient memory terminate operate system android adopt strategy similar ios terminate process insufficient free memory available terminate process android write application state flash memory restriction developer mobile system carefully allocate release memory ensure application use memory suffer memory leak system performance swapping swap page efficient swap entire process system undergo form swapping sign active process available physical memory generally approach handle situation 1 terminate process 2 physical memory example intel 32- 64 bit architecture example intel 32- 64 bit architecture architecture intel chip dominate personal computer landscape decade 16 bit intel 8086 appear late 1970 soon follow 16 bit chip intel 8088 notable chip ia-32 include family 32 bit pentium processor recently intel produce series 64 bit chip base x86 64 architecture currently popular pc operate system run intel chip include windows macos linux linux course run architecture notably intel dominance spread mobile system arm architecture currently enjoy considerable success section 9.7 section examine address translation ia-32 x86 64 architecture proceed important note intel release version variation architecture year provide complete description memory- management structure chip provide cpu detail information well leave book computer architecture present major memory management concept intel cpu memory management ia-32 system divide component segmentation paging work follow cpu generate logical address give segmentation unit segmentation unit produce linear address logical address linear address give paging unit turn generate physical address main memory segmentation paging unit form equivalent memory management unit mmu scheme show figure 9.21 ia-32 architecture allow segment large 4 gb max- imum number segment process 16 k. logical address space process divide partition partition consist 8 k segment private process second partition consist 8 k segment share process information partition keep

partition consist 8 k segment share process information partition keep local descriptor table ldt information second partition keep global descriptor table gdt entry ldt gdt consist 8 byte segment descriptor detailed information particular segment include base location limit segment logical physical address translation ia-32 32 bit linear address logical address pair selector offset selector 16 bit s designate segment number g indicate segment gdt ldt p deal protection offset 32 bit number specify location byte segment question machine segment register allow segment address time process 8 byte microprogram register hold correspond descriptor ldt gdt cache let pentium avoid have read descriptor memory memory reference linear address ia-32 32 bit long form follow segment register point appropriate entry ldt gdt base limit information segment question generate linear address limit check address validity address valid memory fault generate result trap operate system valid value offset add value base result 32 bit linear address show figure 9.22 follow section discuss paging unit turn linear address physical address ia-32 architecture allow page size 4 kb 4 mb 4 kb page ia-32 use level paging scheme division 32 bit linear address follow example intel 32- 64 bit architectures paging ia-32 architecture address translation scheme architecture similar scheme show figure

9.16 ia-32 address translation show detail figure 9.23 10 high order bit reference entry outermost page table ia-32 term page directory cr3 register point page directory current process page directory entry point inner page table index content innermost 10 bit linear address finally low order bit 0–11 refer offset 4 kb page point page table entry page directory page size flag set indicate size page frame 4 mb standard 4 kb flag set page directory point directly 4 mb page frame bypass inner page table 22 low order bit linear address refer offset 4 mb page

low order bit linear address refer offset 4 mb page frame improve efficiency physical memory use ia-32 page table swap disk case invalid bit page directory entry indicate table entry point memory disk table disk operate system use 31 bit specify disk location table table bring memory demand software developer begin discover 4 gb memory limitation 32 bit architecture intel adopt page address extension pae allow 32 bit processor access physical address space large 4 gb fundamental difference introduce pae support paging go level scheme show figure 9.23 level scheme bit refer page directory pointer table figure 9.24 illustrate pae system 4 kb page pae support 2 mb page pae increase page directory page table entry 32 64 bit size allow base address page table page frame 31 30 29 page address extension extend 20 24 bit combine 12 bit offset add pae support ia-32 increase address space 36 bit support 64 gb physical memory important note operating system support require use pae linux macos support pae 32 bit version windows desktop operate system provide support 4 gb physical memory pae enable intel interesting history develop 64 bit architecture ini- tial entry ia-64 later name itanium architecture architec- ture widely adopt chip manufacturer amd begin develop 64 bit architecture know x86 64 base extend exist ia-32 instruction set x86 64 support large logical physical address space architec- tural advance historically amd develop chip base intel architecture role reverse intel adopt amd x86 64 architecture discuss architecture commercial name amd64 intel 64 use general term x86 64 support 64 bit address space yield astonishing 264 byte addressable memory number great 16 quintillion 16 exabyte

64 bit system potentially address mem- ory practice far few 64 bit address representation current design x86 64 architecture currently provide 48 bit virtual address support page

64 architecture currently provide 48 bit virtual address support page size 4 kb 2 mb 1 gb level paging hierarchy representation linear address appear figure 9.25 addressing scheme use pae virtual address 48 bit size support 52 bit physical address 4,096 terabyte x86 64 linear address example armv8 architecture example armv8 architecture intel chip dominate personal computer market 30 year chip mobile device smartphone tablet com- puter instead run arm processor interestingly intel design manufacture chip arm design license architectural design chip manufacturer apple license arm design iphone ipad mobile device android base smart- phone use arm processor addition mobile device arm provide architecture design real time embed system abundance device run arm architecture 100 billion arm processor produce make widely architecture measure quantity chip produce section describe 64 bit armv8 architecture armv8 different translation granule 4 kb 16 kb 64 kb translation granule provide different page size large section contiguous memory know region page region size different translation granule show translation granule size 2 mb 1 gb 4 kb 16 kb granule level paging level paging 64 kb granule figure 9.26 illustrate armv8 address structure 4 kb translation granule level paging notice armv8 64 bit architecture 48 bit currently level hierarchical paging structure 4 kb translation granule illustrate figure 9.27 ttbr register translation table base register point level 0 table current level offset bit 0–11 figure 9.26 refer offset 4 kb page notice table entry level 1 history teach memory capacity cpu speed similar computer capability large satisfy demand foreseeable future growth technology ultimately absorb available capacity find need additional memory processing power soon think future technology bring 64 bit address space small arm 4 kb translation granule level 2 refer table 1 gb region level-1 table 2 mb region level-2 table example level-1

table refer 1 gb region level-2 table low order 30 bit bit 0–29 figure 9.26 offset 1 gb region similarly level-2 table refer 2 mb region level-3 table low order 21 bit bit 0–20 figure 9.26 refer offset 2 mb region arm architecture support level tlb inner level micro tlb tlb datum instruction micro tlb support asid outer level single main tlb address translation begin micro tlb level case miss main tlb check tlb yield miss page table walk perform memory central operation modern computer system consist large array byte address way allocate address space process use base limit register base register hold small legal physical memory address limit specify size range arm level hierarchical paging bind symbolic address reference actual physical address occur 1 compile 2 load 3 execution time address generate cpu know logical address memory management unit mmu translate physical address approach allocate memory allocate partition contiguous memory vary size partition allocate base possible strategy 1 fit 2 good fit 3 bad fit modern operate system use paging manage memory process physical memory divide fix sized block call frame logical memory block size call page paging logical address divide part page number page offset page number serve index per- process page table contain frame physical memory hold page offset specific location frame reference translation

look aside buffer tlb hardware cache page table tlb entry contain page number correspond frame tlb address translation paging system involve obtain page number logical address check frame page tlb frame obtain tlb frame present tlb retrieve page table hierarchical paging involve divide logical address multiple part refer different level page table address expand 32 bit number hierarchical level large strategy address problem hash page table inverted swapping allow system page belong process disk increase degree multiprogramming intel 32 bit architecture level page table support 4 kb 4 mb page size architecture support page- address extension allow 32 bit processor access physical address space large 4 gb x86 64 armv9 architecture 64 bit architecture use hierarchical paging difference logical physical address page size power 2 consider system program separate part code datum cpu know want instruction instruc- tion fetch datum datum fetch store base limit register pair provide instruction datum instruction base limit register pair automatically read program share different user discuss advantage disadvan- tage scheme consider logical address space 64 page 1,024 word map physical memory 32 frame bit logical address bit physical address effect allow entry page table point page frame memory explain effect decrease time need copy large memory place effect update byte page page give memory partition 300 kb 600 kb 350 kb 200 kb 750 kb 125 kb order fit good fit bad fit algorithm place process size 115 kb 500 kb 358 kb 200 kb 375 kb order assume 1 kb page size page number offset following address reference provide decimal number btv operate system 21 bit virtual address certain embed device 16 bit physical address 2 kb page size entry follow conventional single level page table inverted page table maximum physical memory btv operate consider logical address space 256 page 4 kb page

operate consider logical address space 256 page 4 kb page size map physical memory 64 frame bit require logical address bit require physical address

consider computer system 32 bit logical address 4 kb page size system support 512 mb physical memory entry follow conventional single level page table inverted page table concept paging credit designer atlas system describe kilburn et al 1961 howarth et al hennessy patterson 2012 explain hardware aspect tlbs cache mmu jacob mudge 2001 describe technique manage tlb fang et al 2001 evaluate support large page pae support windows systems.i discuss http://msdn.microsoft.co m en library windows hardware gg487512.aspx overview arm architecture provide http://www.arm.com/products/processors/cortex- fang et al 2001 z. fang l. zhang j. b. carter w. c. hsieh s. a. mckee reevaluate online superpage promotion hardware support proceed- ing international symposium high performance computer architecture volume 50 number 5 2001 hennessy patterson 2012 j. hennessy d. patterson computer archi- tecture quantitative approach fifth edition morgan kaufmann 2012 howarth et al 1961 d. j. howarth r. b. payne f. h. sumner manchester university atlas operating system ii

user description computer journal volume 4 number 3 1961 page 226–229 jacob mudge 2001 b. jacob t. mudge uniprocessor virtual mem- ory tlbs ieee transactions computers volume 50 number 5 2001 kilburn et al 1961 t. kilburn d. j. howarth r. b. payne f. h. sumner manchester university atlas operating system internal organiza- tion computer journal volume 4 number 3 1961 page 222–225 chapter 9 exercise explain difference internal external fragmentation consider follow process generate binary compiler generate object code individual module linker combine multiple object module single program binary linker change binding instruction datum mem- ory address information need pass compiler linker facilitate memory bind task linker give memory partition 100 mb 170 mb 40 mb 205 mb 300 mb 185 mb order fit well fit worst fit algorithm place process size 200 mb 15 mb 185 mb 75 mb 175 mb 80 mb order indicate request satisfy comment efficiently algorithm manage system allow program allocate memory address space execution allocation datum heap segment program example allocate memory require support dynamic memory allocation following scheme contiguous memory allocation compare memory organization scheme contiguous memory allo- cation paging respect following issue ability share code process system paging process access memory operate system allow access addi- tional memory explain mobile operate system ios android android support swap boot disk possible set swap space separate sd nonvolatile memory card android disallow swap boot disk allow secondary disk explain address space identifier asid tlb program binary system typically structure follow code store start small fix virtual address 0 code segment follow data segment store program variable program start execute stack allocate end virtual address space allow grow low virtual address significance structure follow scheme contiguous memory allocation assume 1 kb page size page number offset following address reference provide decimal number mpv operate

number offset following address reference provide decimal number mpv operate system

design embed system 24 bit virtual address 20 bit physical address 4 kb page size entry follow conventional single level page table inverted page table maximum physical memory mpv operate consider logical address space 2,048 page 4 kb page size map physical memory 512 frame bit require logical address bit require physical address consider computer system 32 bit logical address 8 kb page size system support 1 gb physical memory entry follow conventional single level page table inverted page table consider paging system page table store memory memory reference take 50 nanosecond long page memory reference add tlbs 75 percent page table reference find tlb effective memory reference time assume find page table entry tlbs take 2 nanosecond entry present purpose page page table consider ia-32 address translation scheme show figure 9.22 describe step take ia-32 translate logical address physical address advantage operate system hardware provide complicated memory translation disadvantage address translation system scheme assume system 32 bit virtual address 4 kb page size write c program pass virtual address decimal command line output page number offset give address example program run follow program output address 19986 contain page number = 4 offset = 3602

write program require appropriate data type store 32 bit encourage use unsigned data type contiguous memory allocation section 9.2 present different algorithm contiguous memory allo- cation project involve manage contiguous region memory size max address range 0 max 1 program respond different request 1 request contiguous block memory 2 release contiguous block memory 3 compact unused hole memory single block 4 report region free allocate memory program pass initial memory startup example following initialize program 1 mb 1,048,576 byte program start present user follow respond following command rq request rl release c compact stat status report x exit request 40,000 byte appear follow allocator > rq p0 40000 w similarly release appear command release memory allocate process p0 command compaction enter command compact unused hole memory region finally stat command report status memory enter give command program report region memory allocate region unused example possible arrange- ment memory allocation follow address 0:315000 process p1 address 315001 512500 process p3 address 512501:625575 unused address 625575:725100

process p6 address 725001 program allocate memory approach high- light section 9.2.2 depend flag pass rq com- mand flag f fit b good fit w bad fit require program track different hole repre- sente available memory request memory arrive allocate memory available hole base allocation strategy insufficient memory allocate request output error message reject request program need track region memory allocate process necessary support stat parameter rq command new process require memory follow memory request finally strategy situation w refer bad fit command need memory release rl command process release memory pass command partition release adjacent exist hole sure combine hole user enter c command program compact set hole large hole example separate hole size 550 kb 375 kb 1,900 kb 4,500 kb program combine hole large hole size 7,325 kb strategy implement compaction suggest section 9.2.3 sure update begin address process affect compaction c h p t e r chapter 9 discuss memory management strategy computer system strategy goal process memory simultaneously allow multiprogramming tend require entire process memory execute virtual memory technique allow execution process completely memory major advantage scheme pro- grams large physical memory virtual memory abstract main memory extremely large uniform array storage separate logical memory view programmer physical memory technique free programmer concern memory storage limita- tion virtual memory allow process share file library implement share memory addition provide efficient mechanism process creation virtual memory easy implement substantially decrease performance carelessly chap- ter provide detailed overview virtual memory examine implement explore complexity benefit define virtual memory describe benefit illustrate page load memory demand paging apply fifo optimal lru page replacement algorithm describe work set process explain relate describe linux windows 10 solaris manage virtual memory design virtual memory manager simulation c programming lan- memory management algorithm outline chapter 9 necessary basic requirement instruction execute physical

algorithm outline chapter 9 necessary basic requirement instruction execute physical memory approach meet requirement place entire logical address space physical memory dynamic linking help ease restriction generally require special precaution extra work programmer requirement instruction physical memory exe- cut necessary reasonable unfortunate limit size program size physical memory fact exami- nation real program show case entire program need instance consider following program code handle unusual error condition error seldom occur practice code execute array list table allocate memory actu- ally need array declare 100 100 element seldom large 10 10 element certain option feature program rarely instance routine u.s. government computer balance budget year case entire program need need time ability execute program partially memory confer benefit program long constrain physical memory available user able write program extremely large virtual address space simplify programming task program physical memory program run time corresponding increase cpu utiliza- tion throughput increase response time turnaround o need load swap portion program memory program run fast run program entirely memory benefit system user virtual memory involve separation logical memory perceive developer physical memory separation allow extremely large virtual memory provide programmer small physical memory available figure 10.1 virtual memory make task programming easy programmer long need worry physical memory available concentrate instead program problem solve virtual address space process refer logical virtual view process store memory typically view process begin certain logical address address 0 exist contiguous memory show figure 10.2 recall chapter 9 fact physical memory organize page frame physical page frame assign process contiguous memory- diagram show virtual memory large physical memory management unit mmu map logical page physical page frame note figure 10.2 allow heap grow upward memory dynamic memory allocation similarly allow stack grow downward memory successive function call large blank space hole heap stack virtual address space require actual physical

space hole heap stack virtual address space require actual physical page heap stack grow virtual address space include hole know sparse address space sparse address space beneficial hole fill stack heap segment grow wish dynamically link library possibly share object program execution virtual address space process memory share library virtual memory addition separate logical memory physical memory virtual memory allow file memory share process page sharing section 9.3.4 lead following benefit system library standard c library share process mapping share object virtual address space process consider library vir- tual address space actual page library reside physical memory share process figure 10.3 typically library map read space process link similarly process share memory recall chapter 3 process communicate use share memory virtual memory allow process create region memory share process process share region consider virtual address space actual physical page memory share illustrate figure 10.3 page share process creation fork system speed process creation explore benefit virtual memory later chapter discuss implement virtual memory 10.2 demand paging consider executable program load secondary storage memory option load entire program physical memory program execution time problem approach initially need entire program memory suppose program start list available option user select load entire program memory result load executable code option regardless option ultimately select alternative strategy load page need tech- nique know demand paging commonly virtual memory system demand page virtual memory page load demand program execution page access load physical memory demand paging system simi- lar paging system swapping section 9.5.2 process reside secondary memory usually hdd nvm device demand paging explain primary benefit virtual memory load portion program need memory efficiently general concept demand paging mention load page memory need result process execute page memory secondary storage need form hardware support distinguish valid invalid bit scheme describe section 9.3.3 purpose page table valid invalid bit scheme describe section 9.3.3 purpose page table page main memory time bit set valid associate page legal memory bit set invalid page valid logical address space process valid currently secondary storage page table entry page bring memory set usual page table entry page currently memory simply mark invalid situation depict figure 10.4 notice mark page invalid effect process attempt access page happen process try access page bring memory access page mark invalid cause page fault page hardware translate address page table notice invalid bit set cause trap operate system trap result operate system failure bring desire page memory procedure handle page fault straightforward figure 10.5 1 check internal table usually keep process control block process determine reference valid invalid memory access 2 reference invalid terminate process valid bring page page 3 find free frame take free frame list example page step handle page fault 4 schedule secondary storage operation read desire page newly allocate frame 5 storage read complete modify internal table keep process page table indicate page memory 6 restart instruction interrupt trap process access page memory extreme case start execute process page memory operate system set instruction pointer instruction process non memory resident page process immediately fault page page bring memory process continue execute fault necessary page need memory point execute fault scheme pure demand paging bring page memory theoretically program access new page memory instruction execution page instruction datum possibly cause multiple page fault instruction situation result unacceptable system performance fortunately analysis run process show behavior exceedingly unlikely program tend locality reference describe section 10.6.1 result reasonable performance demand paging hardware support demand paging hardware paging swapping page table table ability mark entry invalid valid invalid bit special value protection bit secondary memory memory hold page present main memory secondary memory usually high speed

hold page present main memory secondary memory usually high speed disk nvm device know swap device section storage purpose know swap space swap space allocation discuss chapter 11 crucial requirement demand paging ability restart instruction page fault save state register condi- tion code instruction counter interrupted process page fault occur able restart process exactly place state desire page memory accessible case requirement easy meet page fault occur memory reference page fault occur instruction fetch restart fetch instruction page fault occur fetch operand fetch decode instruction fetch bad case example consider address instruction add content b place result c. step execute 1 fetch decode instruction add 2 fetch a. 3

fetch b. 4 add b. 5 store sum c. fault try store c c page currently memory desire page bring correct page table restart instruction restart require fetch instruction decode fetch operand add repeat work complete instruction repetition necessary page fault

major difficulty arise instruction modify dif- character instruction 256 byte location possibly overlap location block source destination straddle page boundary page fault occur par- tially addition source destination block overlap source block modify case simply restart problem solve different way solution microcode compute attempt access end block page fault go occur happen step modify place know page fault occur relevant page memory solution use temporary register hold value overwritten location page fault old value write memory trap occur action restore memory state instruction start instruction repeat means architectural problem result add paging exist architecture allow demand paging illustrate difficulty involve paging add cpu memory computer system entirely transparent process people assume paging add system assumption true non demand paging environment page fault represent fatal error true page fault mean additional page bring memory process restart page fault occur operate system bring desire page secondary storage main memory resolve page fault oper- ating system maintain free frame list pool free frame satisfy request figure 10.6 free frame allocate stack heap segment process expand operate system typically allo- list free frame cate free frame technique know zero fill deman zero fill- demand frame zero allocate erase previous content consider potential security implication clear content frame reassign system start available memory place free frame list free frame request example demand paging size free frame list shrink point list fall zero fall certain threshold point repopulate cover strategy situation section 10.4 performance demand paging demand paging significantly affect performance computer system let compute effective access time demand page mem- ory assume memory access time denote ma 10 nanosecond long page fault effective access time equal memory access time page fault occur read relevant page secondary storage access desire word let p probability page fault 0 p 1

expect p close zero expect page fault effective access time effective access time = 1 p × ma + p × page fault time compute effective access time know time need service page fault page fault cause follow sequence 1 trap operate system 2 save register process state 3 determine interrupt page fault 4 check page reference legal determine location page secondary storage 5 issue read storage free frame a. wait queue read request service b. wait device seek and/or latency time c. begin transfer page free frame 6 wait allocate cpu core process 7 receive interrupt storage o subsystem o complete 8 save register process state process step 6 9 determine interrupt secondary storage device 10 correct page table table desire page memory 11 wait cpu core allocate process 12 restore register process state new page table resume interrupt instruction step necessary case example assume step 6 cpu allocate process o occur arrangement allow multiprogramming maintain cpu utilization require additional time resume page fault service routine o transfer complete case major task component page fault service 1 service page fault interrupt 2 read page 3 restart process task reduce careful coding instruction task 1 100 microsecond let consider case hdd paging device page- switch time probably close 8 millisecond typical hard disk average latency 3 millisecond seek 5 millisecond transfer time 0.05 millisecond total paging time 8 millisecond include hardware software time remember look device service time queue process wait device add queuing time wait page device free service request increase time page average page fault service time 8 millisecond memory- access time 200 nanosecond effective access time nanosecond effective access time = 1 p × 200 + p 8 millisecond = 1 p × 200 + p × 8,000,000 = 200 + 7,999,800 × p.

effective access time directly proportional page fault rate access 1,000 cause page fault effective access time 8.2 microsecond computer slow factor 40 demand paging want performance degradation 10 percent need probability page fault following level 220 > 200 + 7,999,800 × p 20 > 7,999,800 × p p < 0.0000025 slowdown paging

reasonable level allow few memory access 399,990 page fault sum impor- tant page fault rate low demand page system effective access time increase slow process execution dramatically additional aspect demand paging handling overall use swap space o swap space generally fast file system fast swap space allocate large block file lookup indirect allocation method chapter 11 option system gain well paging throughput copy entire file image swap space process startup perform demand paging swap space obvious disadvantage approach copying file image program start second option practice operate system include linux windows demand page file system initially write page swap space replace approach ensure need page read file system subsequent paging swap space system attempt limit swap space demand paging binary executable file demand page file bring directly file system page replacement call frame simply overwrite modify page read file system need approach file system serve backing store swap space page associate file know anonymous memory page include stack heap process method appear good compromise system include linux bsd unix describe section 9.5.3 mobile operate system typically support swapping instead system demand page file sys- tem reclaim read page code application memory constrained datum demand page file system later need ios anonymous memory page reclaim application application terminate explicitly release memory section 10.7 cover compress memory commonly alternative swap mobile system section 10.2 illustrate process start quickly demand- paging page contain instruction process creation fork system initially bypass need demand paging technique similar page sharing cover section 9.3.4 technique provide rapid process creation minimize number new page allocate newly create process recall fork system create child process duplicate parent traditionally fork work create copy parent address space child duplicate page belong parent consider child process invoke exec system immediately creation copying parent address space unnecessary instead use technique know copy write

parent address space unnecessary instead use technique know copy write work allow parent child process initially share page share page mark copy write page mean process write share page copy share page create copy write illustrate figure 10.7 10.8 content physical memory process 1 modify page c. example assume child process attempt modify page contain portion stack page set copy write operate system obtain frame free frame list create copy process 1 modify page c. page map address space child process child process modify copy page page belong parent process obviously copy write technique page modify process copy unmodified page share parent child process note page modify need mark copy write page modify page contain executable code share parent child copy write common technique operate system include windows linux macos version unix include linux macos bsd unix provide variation fork system vfork virtual memory fork operate differently fork copy write vfork parent process suspend child process use address space parent vfork use copy write child process change page parent address space alter page visible parent resume vfork caution ensure child process modify address space parent vfork intend child process call exec immediately creation copying page take place vfork copy page c process 1 modify page c. extremely efficient method process creation implement unix command line shell interface 10.4 page replacement early discussion page fault rate assume page fault reference representation strictly accurate process page actually use half demand paging save o necessary load page increase degree multiprogramming run twice process frame run process run require frame increase degree multiprogramming allocate memory run process page size actually use page high cpu utilization throughput frame spare possible process particular data set suddenly try use page result need frame available consider system memory hold program page buffer o consume considerable memory use increase

hold program page buffer o consume considerable memory use increase strain memory placement algorithm decide memory allocate o program page significant challenge system allocate fix percentage memory o buffer allow process o subsystem compete system memory section 14.6 discuss integrate relationship o buffer virtual memory technique allocation memory manifest follow process execute page fault occur operate system determine desire page reside secondary storage find free frame free frame list memory use situation

illustrate figure 10.9 fact free frame depict question mark operate system option point terminate process demand paging operate system attempt improve computer system utilization throughput user aware process run page system paging logically transparent user option good choice operate system instead use standard swapping swap process free frame reduce level multiprogram- ming discuss section 9.5 standard swapping long operate system overhead copy entire pro- cesse memory swap space operate system com- bine swap page page replacement technique describe detail remainder section basic page replacement page replacement take following approach frame free find currently free free frame write need page replacement content swap space change page table table indicate page long memory figure 10.10 use free frame hold page process fault modify page fault service routine include page replacement 1 find location desire page secondary storage 2 find free frame a. free frame use b. free frame use page replacement algorithm select victim frame c. write victim frame secondary storage necessary change page frame table accordingly 3 read desire page newly free frame change page 4 continue process page fault occur notice frame free page transfer page page require situation effectively double page fault service time increase effective access time accordingly reduce overhead modify bit dirty bit scheme page frame modify bit associate hardware modify bit page set hardware byte page write indicate page modify select page replacement examine modify bit bit set know page modify read secondary storage case write page storage modify bit set page modify read memory case need write memory page storage technique apply read page example page binary code page modify discard desire scheme significantly reduce time require service page fault reduce o time half page modify page replacement basic demand paging complete separation logical memory physical memory mechanism enor- mous virtual memory provide programmer small

physical memory mechanism enor- mous virtual memory provide programmer small physical memory demand paging logical address map physical address set address different page process physical memory demand paging size logical address space long constrain physical memory process page execute frame simply demand paging replacement algorithm find free frame necessary page modify replace content copy secondary storage later reference page cause page fault time page bring memory replace page process solve major problem implement demand paging develop frame allocation algorithm page replacement algorithm multiple process memory decide frame allocate process page replacement require select frame replace design appropriate algo- rithm solve problem important task secondary storage o expensive slight improvement demand paging method yield large gain system performance different page replacement algorithm operate system probably replacement scheme select par- ticular replacement algorithm general want low evaluate algorithm run particular string memory reference compute number page fault string memory reference call reference string generate reference string arti- ficially random number generator example trace give system record address memory reference choice produce large number datum order 1 million address second reduce number datum use fact give page size page size generally fix hard- ware system need consider page number entire address second reference page p reference page p immediately follow cause page fault page p memory reference immediately follow reference fault example trace particular process record follow 0100 0432 0101 0612 0102 0103

0104 0101 0611 0102 0103 0104 0101 0610 0102 0103 0104 0101 0609 0102 0105 100 byte page sequence reduce following reference 1 4 1 6 1 6 1 6 1 6 1 determine number page fault particular reference string page replacement algorithm need know number page frame available obviously number frame available increase number page fault decrease reference string consider previously example frame fault fault reference page contrast frame

consider previously example frame fault fault reference page contrast frame available replacement reference result fault general expect curve figure 10.11 number frame increase number page fault drop minimal level course add physical memory increase number frame use reference string 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1 memory frame fifo page replacement simple page replacement algorithm fifo algo- rithm fifo replacement algorithm associate page time page bring memory page replace old page choose notice strictly necessary record time number page fault number frame graph page fault versus number frame page bring create fifo queue hold page memory replace page head queue page bring memory insert tail queue example reference string frame initially reference 7 0 1 cause page fault bring frame reference 2 replace page 7 page 7 bring 0 reference 0 memory fault reference reference 3 result replacement page 0 line replacement reference 0 fault page 1 replace page 0 process continue show figure 10.12 time fault occur page frame fault altogether fifo page replacement algorithm easy understand program performance good hand page replace initialization module long time ago long need hand contain heavily variable initialize early constant use notice select replacement page active use work correctly replace active page new fifo page replacement algorithm number page fault number frame page fault curve fifo replacement reference string fault occur immediately retrieve active page page replace bring active page memory bad replacement choice increase page fault rate slow process execution cause incorrect execution illustrate problem possible fifo page replacement algorithm consider following reference string 1 2 3 4 1 2 5 1 2 3 4 5 figure 10.13 show curve page fault reference string versus number available frame notice number fault frame great number fault frame unexpected result

notice number fault frame great number fault frame unexpected result know belady anomaly page replacement algorithm page fault rate increase number allocate frame increase expect give memory process improve performance early research investigator notice assumption true belady anomaly discover result optimal page replacement result discovery belady anomaly search optimal page replacement algorithm algorithm low page fault rate algorithm suffer belady anomaly algorithm exist call opt min simply replace page long period time use page replacement algorithm guarantee low possible page- fault rate fix number frame example sample reference string optimal page replacement algorithm yield page fault show figure 10.14 reference cause fault fill frame reference page 2 replace page 7 page 7 reference 18 optimal page replacement algorithm page 0 5 page 1 14 reference page 3 replace page 1 page 1 page memory reference page fault optimal replacement well fifo algorithm result fault ignore algorithm suffer optimal replacement twice good fifo replacement fact replacement algorithm process reference string frame few fault unfortunately optimal page replacement algorithm difficult implement require future knowledge reference string encounter similar situation sjf cpu scheduling algorithm section 5.3.2 result optimal algorithm mainly comparison study instance useful know new algorithm optimal 12.3 percent optimal worst 4.7 percent average lru page replacement optimal algorithm feasible approximation opti- mal algorithm possible key distinction fifo opt algo- rithm look backward versus forward time fifo algorithm use time page bring memory opt algorithm use time page use recent past approximation near future replace page long period time approach recently lru algorithm lru replacement associate page time page use page replace lru choose page long period time think strategy optimal page replacement algorithm look backward time forward strangely let sr reverse reference string s page fault rate opt algorithm s page fault rate opt algorithm sr similarly page fault rate lru fault rate opt algorithm sr similarly page fault rate lru algorithm s page fault rate lru algorithm sr result apply lru replacement example reference string show figure 10.15 lru algorithm produce fault notice fault optimal replacement reference page 4 occur lru replacement see frame memory page 2 recently lru algorithm replace page 2 know page 2 fault lru page replacement algorithm page 2 lru algorithm replace page 3 recently page memory despite problem lru replacement fault well fifo replacement lru policy page replacement algorithm con- sidere good major problem implement lru replacement lru page replacement algorithm require substantial hardware assis- tance problem determine order frame define time use implementation feasible counter simple case associate page table entry time use field add cpu logical clock counter clock incremente memory reference reference page content clock register copy time use field page table entry page way time reference page replace page small time value scheme require search page table find lru page write memory time use field page table memory access time maintain page table change cpu scheduling overflow clock consider stack approach implement lru replacement stack page number page reference remove stack way recently page stack recently page figure 10.16 entry remove middle stack good implement approach doubly link list head pointer tail pointer remove page put stack require change pointer worst update little expensive search replacement tail pointer point stack lru page approach particularly appropriate software microcode implementation lru replacement like optimal replacement lru replacement suffer belady anomaly belong class page replacement algorithm call stack algorithm exhibit belady anomaly stack algorithm algorithm show set page memory n frame subset set page memory n use stack record recent page reference + 1 frame lru replacement set page memory n recently reference page number frame increase n page recently reference memory note implementation

number frame increase n page recently reference memory note implementation lru conceivable hardware assistance standard tlb register updating clock field stack memory reference use interrupt reference allow software update data structure slow memory reference factor slow process factor system tolerate level overhead memory management lru approximation page replacement computer system provide sufficient hardware support true lru page replacement fact system provide hardware support page replacement algorithm fifo algorithm system provide help form reference bit reference bit page set hardware page reference read write byte page reference bit associate entry page table initially bit clear 0 operate system process execute bit associate page reference set 1 hardware time determine page examine reference bit know order use information basis page replacement algorithm approximate lru replacement gain additional ordering information record reference bit regular interval 8 bit byte page table memory regular interval 100 millisecond timer interrupt transfer control operate system operate system shift reference bit page high order bit 8 bit byte shift bit right 1 bit discard low order bit 8 bit shift register contain history page use time period shift register contain 00000000 example page time period page period shift register value 11111111 page history register value 11000100 recently value 01110111 interpret 8 bit byte unsigned integer page low number lru page replace notice number guarantee unique replace swap page small value use fifo method choose number bit history include shift register vary course select depend hardware available updating fast possible extreme case number reduce zero leave reference bit algorithm call second- chance page replacement algorithm basic algorithm second chance replacement fifo replacement algo- rithm page select inspect reference bit value 0 proceed replace page reference bit set 1 page second chance select fifo page page get second chance reference bit clear arrival time reset current time page give second

bit clear arrival time reset current time page give second chance replace page replace give second chance addition page reference bit set replace way implement second chance algorithm refer clock algorithm circular queue pointer hand clock indicate page replace frame need pointer advance find page 0 reference bit advance clear reference bit figure 10.17 victim page find page replace new page insert circular queue position notice bad case bit set pointer cycle queue give page second chance clear reference bit select page replacement second chance replacement degenerate fifo replacement bit set enhanced second chance algorithm enhance second chance algorithm consider reference bit modify bit describe section 10.4.1 order pair bit following possible class 1 0 0 recently modify good page replace 2 0 1 recently modify good page need write replacement 3 1 0 recently clean probably soon circular queue page circular queue page second chance clock page replacement algorithm 4 1 1 recently modify probably soon page need write secondary storage page class page replacement call use scheme clock algorithm instead examine page point reference bit set 1 examine class page belong replace page encounter low nonempty class notice scan circular queue time find page replace major difference algorithm simple clock algorithm preference page modify order reduce number o require counting base page replacement algorithm page replacement example counter number reference page develop follow scheme frequently lfu page replacement algorithm require page small count replace reason selection actively page large reference count problem arise page heavily initial phase process heavily large count remain memory long need solution shift count right 1 bit regular interval form exponentially decay average usage count frequently mfu page replacement algorithm base argument page small count probably bring expect mfu lfu replacement common imple- mentation algorithm expensive approximate opt procedure addition specific page replacement algorithm

algorithm expensive approximate opt procedure addition specific page replacement algorithm example system commonly pool free frame page fault occur victim frame choose desire page read free frame pool victim write procedure allow process restart soon possible wait victim page write victim later write frame add free frame pool expansion idea maintain list modify page page device idle modify page select write secondary storage modify bit reset scheme increase probability page clean select replacement need modification pool free frame remember page frame frame content modify frame write secondary storage old page reuse directly free frame pool need frame reuse o need case page fault occur check desire page free frame pool select free frame read version unix system use method conjunction second chance algorithm useful augmentation page- replacement algorithm reduce penalty incur wrong victim page select describe modification section 10.5.3 application page replacement certain case application access datum operate system vir- tual memory perform worse operate system provide buffer- ing typical example database provide memory management o buffering application like understand mem- ory use storage use well operate system implement- e algorithm general purpose use furthermore operate system set o. example data warehouse frequently perform massive sequen- tial storage read follow computation write lru algorithm allocation frames remove old page preserve new one applica- tion likely read old page new one start sequential read mfu actually efficient lru problem operating system special program ability use secondary storage partition large sequential array logical block file system data structure array some- time call raw disk o array term raw o. raw o bypass file system service file o demand paging file locking prefetching space allocation file name directory note certain application efficient implement special purpose storage service raw partition application perform well use regular file system service 10.5 allocation frame turn issue allocation

regular file system service 10.5 allocation frame turn issue allocation allocate fix free memory process 93 free frame process frame process consider simple case system 128 frame operate system 35 leave 93 frame user process pure demand paging 93 frame initially free frame list user process start execution generate sequence page fault 93 page fault free frame free frame list free frame list exhaust page replacement algorithm select 93 memory page replace 94th process terminate 93 frame place free frame list variation simple strategy require operate system allocate buffer table space free frame list space use operate system support user paging try free frame reserve free frame list time page fault occur free frame available page page swap take place replacement select write storage device user process continue execute variant possible basic strategy clear user process allocate free frame minimum number frame strategy allocation frame constrain way example allocate total number available frame page sharing allocate minimum number frame look closely requirement reason allocate minimum number frame involve performance obviously number frame allocate process decrease page fault rate increase slow process execution addi- tion remember page fault occur execute instruction complete instruction restart consequently frame hold different page single instruction example consider machine memory reference instruc- tion reference memory address case need frame instruction frame memory reference addition level indirect addressing allow example load instruction frame 16 refer address frame 0 indirect reference frame 23 paging require frame process think happen process frame minimum number frame define computer architecture example instruction give architecture include word addressing mode instruction straddle frame addition operand indirect reference total frame require example instruction intel 32- 64 bit architecture allow datum register register register memory allow direct memory- memory movement limit require minimum number frame process minimum number frame process define architecture maximum number define

process minimum number frame process define architecture maximum number define available physical memory leave significant choice frame easy way split m frame n process equal share m n frame ignore frame need operate system moment instance 93 frame 5 process process 18 frame 3 leftover frame free frame buffer pool scheme call equal allocation alternative recognize process need differing amount memory consider system 1 kb frame size small student process 10 kb interactive database 127 kb process run system 62 free frame sense process 31 frame student process need 10 frame 21 strictly speak waste solve problem use proportional allocation allocate available memory process accord size let size virtual memory process pi si define s = si total number available frame m allocate ai frame process pi ai approximately ai = si s × m. course adjust ai integer great minimum number frame require instruction set sum proportional allocation split 62 frame pro- cesse 10 page 127 page allocate 4 frame 57 frame respectively allocation frame 10/137 × 62 4 127/137 × 62 57 way process share available frame accord need equally equal proportional allocation course allocation vary accord

multiprogramming level multiprogramming level increase process lose frame provide memory need new process conversely multiprogramming level decrease frame allocate depart process spread notice equal proportional allocation high priority process treat low priority process definition want high priority process memory speed execution detriment low priority process solution use proportional allocation scheme ratio frame depend relative size process priority process combination size priority global versus local allocation important factor way frame allocate pro- cesse page replacement multiple process compete frame classify page replacement algorithm broad category global replacement local replacement global replacement allow process select replacement frame set frame frame currently allocate process process frame local replacement require process select set allocate frame example consider allocation scheme allow high- priority process select frame low priority process replacement process select replacement frame frame low priority process approach allow high priority process increase frame allocation expense low priority process local replacement strategy number frame allocate process change global replacement process happen select frame allocate process increase number frame allocate assume process choose frame problem global replacement algorithm set page memory process depend paging behavior pro- cess paging behavior process process perform differently example take 0.5 second execution 4.3 second execution totally external circumstance case local replacement algorithm local replacement set page memory process affect paging behavior process local replacement hinder pro- cess make available page memory global replacement generally result great system throughput commonly method major minor page fault describe section 10.2.1 page fault occur page valid mapping address space process operate system generally distinguish type page fault major minor fault windows refer major minor fault hard soft fault respectively major page fault occur page reference page memory service major page fault require read desire page backing store free frame update page table demand paging typically generate initially high rate major page minor page

paging typically generate initially high rate major page minor page fault occur process logical mapping page page memory minor fault occur reason process reference share library memory process mapping page table instance necessary update page table refer exist page memory second cause minor fault occur page reclaim process place free frame list page zero allocate process kind fault occur frame remove free frame list reassign process expect resolve minor page fault typically time consume resolve major page fault observe number major minor page fault linux system command ps -eo min flt maj flt cmd output number minor major page fault command launch process example output ps command appear interesting note command number major page fault generally low number minor fault high indicate linux process likely significant advantage share library library load memory subsequent page fault minor fault focus possible strategy use implement global page replacement policy approach satisfy memory request free frame list wait list drop zero begin select page replacement trigger page replace- ment list fall certain threshold strategy attempt ensure sufficient free memory satisfy new request strategy

depict figure 10.18 strategy purpose free memory minimum threshold drop allocation frames threshold kernel routine trigger begin reclaim page process system typically exclude kernel kernel routine know reaper apply page- replacement algorithm cover section 10.4 free memory reach maximum threshold reaper routine suspend resume free memory fall minimum threshold figure 10.18 point free memory drop minimum threshold kernel begin reclaim page add free frame list continue maximum threshold reach point b time additional request memory point c free memory fall minimum threshold page reclamation resume suspend free memory reach maximum threshold point d process continue long system run mention kernel reaper routine adopt page- replacement algorithm typically use form lru approximation consider happen reaper routine unable maintain list free frame minimum threshold circum- stance reaper routine begin reclaim page aggressively example suspend second chance algorithm use pure fifo extreme example occur linux free memory fall low level routine know memory oom killer select process terminate free memory linux determine process terminate process know oom score high score increase likelihood process terminate oom killer routine oom score calcu- late accord percentage memory process high percentage high oom score oom score view /proc file system score process pid 2500 view /proc/2500 oom score general reaper routine vary aggressively reclaim memory value minimum maximum threshold vary value set default value system allow system administrator configure base physical memory system non uniform memory access far coverage virtual memory assume main memory create equal access equally non- uniform memory access numa system multiple cpu section 1.3.2 case system give cpu access section main memory fast access performance difference cause cpu memory interconnect system system multiple cpu local memory figure 10.19 cpu organize shared system interconnect expect cpu access local memory fast memory local cpu numa system exception slow system access main memory treat equally describe

system exception slow system access main memory treat equally describe section 1.3.2 numa system accommodate cpu achieve great level throughput parallelism numa multiprocessing architecture manage page frame store location signifi- cantly affect performance numa system treat memory uniform system cpu wait significantly long memory access modify memory allocation algorithm numa account describe modification section 5.5.4 goal memory frame allocate close possible cpu process run definition close minimum latency typically mean system board cpu process incur page fault numa aware virtual memory system allocate process frame close possible cpu process run numa account scheduler track cpu process run scheduler try schedule process previous cpu virtual memory system try allocate frame process close cpu schedule improved cache hit decrease memory access time result picture complicated thread add example process run thread end thread schedule different system board memory allocate discuss section 5.7.1 linux manage situation have kernel identify hierarchy scheduling domain linux cfs scheduler allow thread migrate different domain incur memory access penalty linux separate free frame list numa node ensure thread allocate memory node run solaris solve problem similarly create lgroup locality group kernel lgroup gather cpu memory cpu group access memory group define latency interval addition hierarchy lgroup base latency group similar hierarchy scheduling domain linux solaris try schedule thread process allocate memory process lgroup possible pick nearby lgroup rest resource need practice minimize overall memory latency maximize cpu cache hit rate consider occur process frame minimum number frame need support page work set process quickly page fault point replace page page active use replace page need right away consequently quickly fault replace page bring immediately high paging activity call thrashing process thrash spend time page execute expect thrashing result severe performance problem cause thrashing consider follow scenario base actual behavior early paging system operate system monitor cpu utilization cpu

behavior early paging system operate system monitor cpu utilization cpu utiliza- tion low increase degree multiprogramming introduce new process system global page replacement algorithm replace page regard process belong suppose process enter new phase execution need frame start fault take frame away process process need page fault take frame process fault process use page device swap page queue page device ready queue empty process wait page device cpu utilization decrease cpu scheduler see decrease cpu utilization increase degree multiprogramming result new process try start take frame run process cause page fault long queue page device result cpu utilization drop cpu scheduler try increase degree multiprogramming thrash occur system throughput plunge page- fault rate increase tremendously result effective memory access time increase work getting process spend phenomenon illustrate figure 10.20 cpu utilization plot degree multiprogramming degree multi- programming increase cpu utilization increase slowly maximum reach degree multiprogramming increase thrashing set cpu utilization drop sharply point increase cpu utilization stop thrash decrease degree limit effect thrashing local replacement algo- rithm priority replacement algorithm mention early local replace- ment require process select set allocate frame process start thrash steal frame pro- cess cause thrash problem entirely solve process thrash queue paging device time average service time page fault increase degree multiprogramming long average queue paging device effective access time increase process thrash prevent thrashing provide process frame need know frame need strategy start look frame process actually approach define locality model process execution locality model state process execute move locality locality locality set page actively run program generally compose different locality over- lap example function call define new locality locality memory reference pattern page reference table 2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 1

4 4 4 3 4 3 4 4 4 1 3 2 3 4 4 4 3 4 4 4 ws(t1 = 1,2,5,6,7 ws(t2 = 3,4 locality memory reference instruction function local variable subset global variable exit function process leave locality local variable instruction function long active use return locality later figure 10.21 illustrate concept locality process locality change time time locality set page 18 19 20 21 22 23 24 29 30 33 18 19 20 24 25 26 27 28 29 31 32 33 notice overlap page example 18 19 20 locality locality define program structure data structure locality model state program exhibit basic memory reference structure note locality model unstated principle caching discussion far book access type datum random patterned caching useless suppose allocate frame process accommodate cur- rent locality fault page locality page memory fault change locality allocate frame accommodate size current locality process thrash memory page work set model base assumption locality model use parameter δ

define work set window idea examine recent δ page reference set page recent δ page reference work set figure 10.22 page active use work set long drop work set δ time unit reference work set approximation program locality example give sequence memory reference show figure 10.22 δ = 10 memory reference work set time t1 1 2 5 6 7 time t2 work set change 3 4 accuracy work set depend selection δ. δ small encompass entire locality δ large overlap locality extreme δ infinite

working set set page touch process execution important property work set size compute work set size wssi process system consider d = wssi d total demand frame process actively page work set process need wssi frame total demand great total number available frame d > m thrashing occur process frame δ select use work set model simple operate system monitor work set process allocate work set frame provide work set size extra frame process initiate sum work set size increase exceed total number available frame operate system select process suspend process page write swap frame reallocate process suspend process restart later work set strategy prevent thrashing keep degree multiprogramming high possible optimize cpu utilization difficulty work set model keep track working set working set page fault rate direct relationship work set process page fault rate typically show figure 10.22 working set process change time reference datum code section locality assume sufficient memory store working set process process thrash page fault rate process transition peak valley time general behavior show apeak page fault rate occur begin demand page new locality working set new locality memory page fault rate fall process move new working set page- fault rate rise peak return low rate new working set load memory span time start peak start peak represent transition working set work set window move window memory reference new reference appear end old reference drop end page work set reference work set approximate work set model fix interval timer interrupt reference bit example assume δ equal 10,000 ref- erence cause timer interrupt 5,000 reference timer interrupt copy clear reference bit value page page fault occur examine current reference bit memory bit determine page 10,000 15,000 reference bit bit page bit consider work set note arrangement entirely accurate tell interval 5,000 reference occur reduce uncertainty increase entirely accurate tell interval 5,000 reference occur reduce uncertainty increase number history bit frequency inter- rupt example 10 bit interrupt 1,000 reference cost service frequent interrupt correspondingly high work set model successful knowledge work set useful prepaging section 10.9.1 clumsy way control thrashing strategy use page fault frequency pff take specific problem prevent thrashing thrashing high page fault rate want control page fault rate high know process need frame conversely page- fault rate low process frame establish upper low bound desire page fault rate figure 10.23 actual page fault rate exceed upper limit allocate process number frame frame page fault rate fall low limit remove frame process directly measure control page- fault rate prevent thrashing work set strategy swap process page fault rate increase free frame available select process swap backing store free frame distribute process high page fault rate practically speak thrashing result swapping disagreeably high impact performance current good practice implement computer system include physical memory possible avoid thrashing swapping smartphone large server provide memory work set memory concurrently extreme condition provide good user experience

10.7 memory compression alternative paging memory compression page modify frame swap space compress frame single frame enable system reduce memory usage resort figure 10.24 free frame list contain frame assume number free frame fall certain threshold trigger page replace- ment replacement algorithm lruapproximation algorithm select frames—15 3 35 26 place free frame list place frame modify frame list typically modify frame list write swap space make frame available free frame list alternative strategy compress number frame store compress version single page frame figure 10.25 frame 7 remove free frame list frame 15 3 35

compress store frame 7 store list compress frame frame 15 3 35 move free frame list compress frame later reference page fault occur compress frame decompress restore page 15 3 35 memory modify frame list free frame list compression modify frame list compress frame list free frame list compression note mobile system generally support stan- dard swapping swapping page memory compression integral memory management strategy mobile operate system include android ios addition windows 10 macos support memory compression windows 10 microsoft develop universal windows platform uwp architecture provide common app plat- form device run windows 10 include mobile device uwp app run mobile device candidate memory compression macos support memory compression version 10.9 operate sys- tem compress lru page free memory short page solve problem performance test indicate memory com- pression fast page ssd secondary storage laptop desktop macos system memory compression require allocate free frame hold compress page significant memory saving realize depend reduction achieve compression algorithm example frame reduce original size form data compression contention speed compression algorithm reduction achieve know compression ratio general high compression ratio great reduction achieve slow computationally expensive algorithm algorithm use today balance factor achieve relatively high compression ratio fast algorithm addition compression algorithm improve take advantage multiple com- put core perform compression parallel example microsoft xpress apple wkdm compression algorithm consider fast report compress page 30 50 percent original size 10.8 allocate kernel memory process run user mode request additional memory page allocate list free page frame maintain kernel list typically populate page replacement algorithm dis- cuss section 10.4 likely contain free page scatter physical memory explain early remember user process request single byte memory internal fragmentation result process grant entire page frame allocate kernel memory kernel memory allocate free memory pool different list satisfy ordinary user mode process primary reason 1

different list satisfy ordinary user mode process primary reason 1 kernel request memory data structure vary size page size result kernel use memory conservatively attempt minimize waste fragmentation especially important operate system subject kernel code datum page system 2 page allocate user mode process necessarily contiguous physical memory certain hardware device interact directly physical memory benefit virtual memory interface consequently require memory reside physically follow section examine strategy manage free memory assign kernel process buddy system slab allocation buddy system allocate memory fix size segment consist physically contiguous page memory allocate segment power of-2 allocator satisfy request unit size power 2 4 kb 8 kb 16 kb forth request unit appropriately size round high power 2 example request 11 kb satisfy 16 kb segment let consider simple example assume size memory segment initially 256 kb kernel request 21 kb memory segment initially divide buddy al ar 128 kb size buddy divide 64 kb buddy bl br high power 2 21 kb 32 kb bl br divide 32 kb buddy cl cr buddy satisfy 21 kb request scheme

illustrate figure 10.26 cl segment allocate 21 kb request advantage buddy system quickly adjacent buddy combine form large segment technique know coalesce figure 10.26 example kernel release cl unit allocate system coalesce cl cr 64 kb segment segment bl turn coalesce buddy br form 128 kb segment ultimately end original 256 kb segment obvious drawback buddy system round high power 2 likely cause fragmentation allocate seg- ment example 33 kb request satisfy 64 kb segment fact guarantee 50 percent allocate unit waste internal fragmentation follow section explore memory allocation scheme space lose fragmentation second strategy allocate kernel memory know slab allocation slab physically contiguous page cache consist physically contiguous page buddy system allocation slab single cache unique kernel datum struc- ture example separate cache datum structure represent process descriptor separate cache file object separate cache semaphore forth cache populate object instantiation kernel datum structure cache represent example cache represent- e semaphore store instance semaphore object cache represent process descriptor store instance process descriptor object forth relationship slab cache object show figure 10.27 figure show kernel object 3 kb size object 7 kb size store separate cache allocate kernel memory slab allocation algorithm use cache store kernel object cache create number object initially mark free allocate cache number object cache depend size associate slab example 12 kb slab contiguous 4 kb page store 2 kb object initially object cache mark free new object kernel datum structure need allocator assign free object cache satisfy request object assign cache mark let consider scenario kernel request memory slab allocator object represent process descriptor linux system process descriptor type struct task struct require approximately 1.7 kb memory linux kernel create new task request necessary memory struct task struct object cache cache fulfill request struct task struct object allocate slab mark free linux slab possible

task struct object allocate slab mark free linux slab possible state 1 object slab mark 2 object slab mark free 3 partial slab consist free object slab allocator attempt satisfy request free object partial slab exist free object assign slab slab available new slab allocate contiguous physical page assign cache memory object allocate slab allocator provide main benefit 1 memory waste fragmentation fragmentation issue unique kernel data structure associate cache cache slab divide chunk size object represent kernel request memory object slab allocator return exact memory require represent object 2 memory request satisfy quickly slab allocation scheme particularly effective manage memory object fre- quently allocate deallocate case request kernel act allocate release memory time- consume process object create advance quickly allocate cache furthermore kernel finish object release mark free return cache make immediately available subsequent request kernel slab allocator appear solaris 2.4 kernel general purpose nature allocator certain user mode memory request solaris linux originally buddy system begin version 2.2 linux kernel adopt slab allocator linux refer slab implementation slab recent distribution linux include kernel memory allocator slob slub allocator slob allocator design system limited mem- ory embed system slob stand simple list block maintain list object small object 256 byte medium object 1,024 byte large object size page memory request allocate object appropri- ate list fit policy begin version 2.6.24 slub allocator replace slab default allocator linux kernel slub reduce overhead require slab allocator instance slab store certain meta- datum slab slub store datum page structure linux kernel use page additionally slub include cpu queue slab allocator maintain object cache system large number processor memory allocate queue significant slub provide well performance number processor system increase 10.9 consideration major decision paging system selection replacement algorithm allocation policy discuss early chapter consideration discuss obvious property pure demand paging large number

chapter consideration discuss obvious property pure demand paging large number page fault occur process start situation result try initial locality memory prepaging attempt prevent high level initial paging strategy bring page need memory time system work set model example process list page working set suspend process lack free frame remember working set process process resume o finish free frame available automatically bring memory entire working set restart process prepaging offer advantage case question simply cost prepaging cost service correspond page fault case page bring memory prepaging assume s page prepage fraction α s page actually 0 α 1 question cost s α save page fault great cost prepage s 1 α unnecessary page α close 0 prepaging lose α close 1 prepaging win note prepage executable program difficult unclear exactly page bring prepage file predictable file access sequentially linux readahead system prefetche content file memory subsequent access file place main memory designer operate system exist machine seldom choice concern page size new machine design decision good page size

expect single good page size set factor support size page size invariably power 2 generally range 4,096 212 4,194,304 222 byte select page size concern size page table give virtual memory space decrease page size increase number page size page table virtual memory 4 mb 222 example 4,096 page 1,024 byte 512 page 8,192 byte active process copy page table large page size desirable memory well utilize small page process allocate memory start location 00000 continue need probably end exactly page boundary final page allocate page unit allocation unused create internal fragmentation assume independence process size page size expect average half final page process waste loss 256 byte page 512 byte 4,096 byte page

8,192 byte minimize internal fragmentation need small page size

problem time require read write page section 11.1 storage device hdd o time compose seek latency transfer time transfer time proportional transfer page size)—a fact argue small page size latency seek time normally dwarf transfer time transfer rate 50 mb second take 0.01 millisecond transfer 512 byte latency time 3 millisecond seek time 5 millisecond total o time 8.01 millisecond 0.1 percent attributable actual transfer double page size increase o time 8.02 millisecond take 8.02 millisecond read single page 1,024 byte 16.02 millisecond read page 512 byte desire minimize o time argue large small page size total o reduce locality improve small page size allow page match program locality accurately example consider process 200 kb size half 100 kb actually execution large page bring entire page total 200 kb transfer allocate instead page 1 byte bring 100 kb actually result 100 kb transfer allocate small page size well resolution allow isolate memory actually need large page size allocate transfer need happen page need small page size result o total allocate memory notice page size 1 byte page fault byte process 200 kb half memory generate page fault page size 200 kb 102,400 page fault page size 1 byte page fault generate large overhead need process interrupt save register replace page queue page device update table minimize number page fault need large page size factor consider relationship page size sector size page device problem good answer see factor internal fragmentation locality argue small page size table size o time argue large page size historical trend large page size mobile system edition operating system concepts 1983 4,096 byte upper bound page size value common page size 1990 modern system use large page size follow section chapter 9 introduce hit ratio tlb recall hit ratio tlb refer percentage virtual address translation resolve tlb

hit ratio tlb refer percentage virtual address translation resolve tlb page table clearly hit ratio relate number entry tlb way increase hit ratio increase number entry come cheaply associative memory construct tlb expensive power related hit ratio similar metric tlb reach tlb reach refer memory accessible tlb simply number entry multiply page size ideally working set process store tlb process spend considerable time resolve memory reference page table tlb double number entry tlb double tlb reach memory intensive application prove insufficient store work set approach increase tlb reach increase size page provide multiple page size increase page size 4 kb 16 kb quadruple tlb reach lead increase fragmentation application require large page size alternatively architecture provide support page size operate system configure advantage support example default page size linux system 4 kb linux provide huge page feature designate region physical memory large page example 2 mb recall section 9.7 armv8 architecture provide support page region different size additionally tlb entry armv8 contain contiguous bit bit set particular tlb entry entry map contiguous adjacent block memory possible arrangement contiguous block map single tlb entry increase 1 64 kb tlb entry comprise 16 × 4 kb adjacent block 2 1 gb tlb entry comprise 32 × 32 mb adjacent block 3 2 mb tlb entry comprise 32 × 64 kb adjacent block 128 × 16 kb adjacent block provide support multiple page size require operate sys- tem hardware manage tlb example field tlb entry indicate size page frame correspond entry case arm architecture indicate entry refer contiguous block memory manage tlb software hard- ware come cost performance increase hit ratio tlb reach offset performance cost inverted page tables section 9.4.3 introduce concept inverted page table purpose form page management reduce physical memory need track virtual physical address translation accomplish saving create table

need track virtual physical address translation accomplish saving create table entry page physical memory index pair < process id page number > information virtual memory page store physical frame inverted page table reduce physical memory need store information inverted page table long contain complete information logical address space process information require reference page currently memory demand paging require information process page fault information available external page table process keep table look like traditional process page table contain information virtual page locate external page table negate utility inverted page table table reference page fault occur need available quickly instead page memory necessary unfortunately page fault cause virtual memory manager generate page fault page external page table need locate virtual page back store special case require careful handling kernel delay page lookup processing demand paging design transparent user program case user completely unaware page nature memory case system performance improve user compiler awareness underlie demand paging let look contrived informative example assume page 128 word size consider c program function initialize 0 element 128 by-128 array following code typical int j j = 0 j < 128 j++ = 0 < 128 i++ data[i][j = 0 notice array store row major array store data[0][0 data[0][1 data[0][127 data[1][0 data[1][1 data[127][127 page 128 word row take page precede code zero word page word page operate system allocate few 128 frame entire program execution result 128 × 128 = 16,384 page fault contrast suppose change code int j = 0 < 128 i++ j = 0 j < 128 j++ data[i][j = 0 code zero word page start page reduce number page fault 128 careful selection datum structure programming structure increase locality lower page fault rate number page work set example stack good locality access hash table contrast design scatter reference produce bad locality course

hash table contrast design scatter reference produce bad locality course locality reference measure efficiency use datum structure heavily weight factor include search speed total number memory reference total number page touch later stage compiler loader significant effect paging separate code datum generate reentrant code mean code page read modify clean page page replace loader avoid place routine page boundary keep routine completely page routine time pack page packaging variant bin packing problem operation research try pack variable sized load segment fix sized page interpage reference minimize approach particularly useful large page size o interlock page locking demand paging need allow page lock memory situation occur o user virtual memory o implement separate o processor example controller usb storage device generally give number byte transfer memory address buffer figure 10.28 transfer complete cpu interrupt sure following sequence event occur aprocess issue o request queue o device cpu give process process cause page fault global replacement algorithm replace page contain memory buffer wait process page page time later o request advance head device queue o occur specify address frame different page belong process common solution problem solution execute o user memory instead datum copy system memory user memory o take place system memory o device write block tape copy block system memory write tape extra copying result unacceptably high overhead solution allow page lock memory lock bit associate frame frame lock select replacement approach write block disk lock memory page contain block system continue usual locked page replace o complete page lock bit situation frequently operate system kernel lock memory operate system can- tolerate page fault cause kernel specific kernel module include perform memory management user process need lock page memory database process want manage chunk memory example move block secondary storage reason frame o memory memory good knowledge go use datum

reason frame o memory memory good knowledge go use datum pinning page memory fairly common operate system system allow application request region logical address space pin note feature abuse cause stress memory management algorithm application frequently require special privilege request use lock bit involve normal page replacement consider following sequence event low priority process fault select replacement frame page system read necessary page memory ready continue low priority process enter ready queue wait cpu low priority process select cpu scheduler time low priority process wait high priority process fault look replacement page system see page memory reference modify page low priority process bring page look like perfect replacement clean need write apparently long time high priority process able replace low- priority process policy decision simply delay low priority process benefit high priority process waste effort spend bring page low priority process decide prevent replacement newly bring page use lock bit implement mechanism page select replacement lock bit turn remain fault process dispatch lock bit dangerous lock bit turn turn situation occur bug operate system example lock frame unusable instance solaris allow locking hint free disregard hint free frame pool small individual process request page lock memory 10.10 operating system example section describe linux windows solaris manage virtual section 10.8.2 discuss linux manage kernel memory slab allocation cover linux manage virtual memory linux use demand paging allocate page list free frame addition use global page replacement policy similar lru approximation clock algo- rithm describe section 10.4.5.2 manage memory linux maintain type page list active list inactive list active list contain page consider use inactive list con- tain page recently reference eligible linux active list inactive list structure page access bit set page reference actual bit mark page access vary architecture page allocate access bit set add rear active list similarly page

allocate access bit set add rear active list similarly page active list reference access bit set page move rear list periodically access bit page active list reset time recently page active list migrate rear inactive list page inactive list reference move rear active list pattern illustrate figure 10.29 list keep relative balance active list grow large inactive list page active list inactive list eligible reclamation linux kernel page daemon process kswapd periodically awak- en check free memory system free memory fall certain threshold kswapd begin scan page inac- tive list reclaim free list linux virtual memory man- agement discuss great detail chapter 20 windows 10 support 32- 64 bit system run intel ia-32 x86-

64 arm architecture 32 bit system default virtual address space process 2 gb extend 3 gb 32 bit system support 4 gb physical memory 64 bit system windows 10 128 tb vir- tual address space support 24 tb physical memory version windows server support 128 tb physical memory windows 10 imple- ment memory management feature describe far include share library demand paging copy write paging memory com- windows 10 implement virtual memory demand paging clus- tering strategy recognize locality memory reference handle page fault

bring faulting page page immediately precede follow faulting page size cluster vary page type data page cluster contain pages(the page page faulting page page fault cluster size seven key component virtual memory management windows 10

work set management process create assign working- set minimum 50 page work set maximum 345 page work set minimum minimum number page process guar- ante memory sufficient memory available process assign page work set maximum process con- figure hard work set limit value ignore process grow work set maximum sufficient memory available similarly memory allocate process shrink minimum period high demand memory windows use lru approximation clock algorithm describe sec- tion 10.4.5.2 combination local global page replacement policy virtual memory manager maintain list free page frame associate list threshold value indicate sufficient free memory available page fault occur process working- set maximum virtual memory manager allocate page list free page process work set maximum incur page fault sufficient memory available process allocate free page allow grow work set maximum free mem- ory insufficient kernel select page process working set replacement local lru page replacement policy free memory fall threshold vir- tual memory manager use global replacement tactic know automatic work set trimming restore value level threshold automatic work set trimming work evaluate number page allocate process process allocate page work set minimum virtual memory manager remove page work set sufficient memory available process reach work set minimum large process idle target small active process trimming procedure continue sufficient free memory necessary remove page process work set minimum windows perform work set trimming user mode system process solaris thread incur page fault kernel assign page fault thread list free page maintain imperative kernel sufficient free memory available associate list free page parameter lotsfree repre- sent threshold begin page lotsfree parameter typically set 164 size physical memory time second kernel check free memory lotsfree number free page fall lotsfree process know pageout start pageout process similar second chance algorithm describe section 10.4.5.2 use hand scan page pageout process work follow hand clock

use hand scan page pageout process work follow hand clock scan page memory set reference bit 0 later hand clock examine reference bit page memory append page reference bit set 0

free list write content secondary storage modify solaris manage minor page fault allow process reclaim page free list page access reassign process pageout algorithm use parameter control rate page scan know scanrate scanrate express page second range slowscan fastscan free memory fall lotsfree scanning occur slowscan page second progress fastscan depend free memory available default value slowscan 100 page second fastscan typically set value total physical pages)/2 page second maximum 8,192 page second show figure 10.30 fastscan set distance page hand clock determine system parameter handspread time hand clear bit hand investigate value depend scanrate handspread scanrate 100 page second handspread 1,024 page 10 second pass time bit set hand time check hand demand place memory system scanrate thousand uncommon mean time clear investigate bit second free memory solaris page scanner mention pageout process check memory time second free memory fall value desfree desire free memory system pageout run time second intention keep desfree free memory available figure 10.30 pageout process unable free memory desfree 30 second average kernel begin swap process free page allocate swap process general kernel look process idle long period time system unable maintain free memory minfree pageout process call request new page page scan algorithm skip page belong library share process eligible claim scanner algorithm distinguish page allocate process page allocate regular datum file know priority paging cover section 14.6.2 virtual memory abstract physical memory extremely large uni- form array storage benefit virtual memory include follow 1 program large physical memory 2 program need entirely memory 3 process share memory 4 process create demand paging technique page load demand program execution page demand load memory page fault occur page currently memory access page bring backing store avail- able page frame memory copy write allow child process share address space parent child parent process write modify

process share address space parent child parent process write modify page copy page available memory run low page replacement algorithm select exist page memory replace new page page- replacement algorithm include fifo optimal lru pure lru algorithm impractical implement system instead use global page replacement algorithm select page process system replacement local page replacement algorithm select page faulting process thrashing occur system spend time page execute locality represent set page actively process execute move locality locality work set base locality define set page currently use process memory compression memory management technique com- press number page single page compressed memory alternative paging mobile system support kernel memory allocate differently user mode process allo- cat contiguous chunk vary size common technique allocate kernel memory 1 buddy system 2 slab allocation tlb reach refer memory accessible tlb equal number entry tlb multiply page size technique increase tlb reach increase size page linux windows solaris manage virtual memory similarly demand paging copy write feature system use variation lru approximation know clock algorithm circumstance page fault occur describe action take operate system page fault occur assume page reference string process m frame initially page reference string length p n distinct page number occur answer question low bound number page fault upper bound number page fault consider follow page replacement algorithm rank algo- rithm point scale bad perfect accord page fault rate separate algorithm suffer belady anomaly operate system support page virtual memory central processor cycle time 1 microsecond cost additional 1 microsecond access page current page 1,000 word page device drum rotate 3,000 revolution minute transfer 1 million word second following statistical measurement obtain system percent instruction execute access page current page instruction access page 80 percent access page memory new page require replace page modify 50 percent time calculate effective instruction time system assume modify 50 percent time calculate effective instruction time system assume system run process processor idle drum transfer consider page table system 12 bit virtual physical address 256 byte page list free page frame d e f d head list e second f dash page frame indicate page memory convert following virtual address equivalent physical address hexadecimal number give hexadecimal discuss hardware function require support demand paging consider dimensional array int = new int[100][100 a[0][0

location 200 page memory system page size 200 small process manipulate matrix reside page 0 location 0 199 instruction fetch page 0 page frame page fault generate following array initialization loop use lru replacement assume page frame 1 contain process initially int j = 0 j < 100 j++ int = 0 < 100 i++ a[i][j = 0 int = 0 < 100 i++ int j = 0 j < 100 j++ a[i][j = 0 consider following page reference string 1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6 page fault occur following replacement algorithm assume seven frame remember frame initially unique page cost fault lru replacement fifo replacement optimal replacement consider following page reference string 7 2 3 1 2 5 3 4 6 7 7 1 0 5 4 6 2 3 0 1

assume demand paging frame page fault occur following replacement algorithm lru replacement fifo replacement optimal replacement suppose want use paging algorithm require ref- erence bit second chance replacement work set model hardware provide sketch simu- late reference bit provide hardware explain possible possible calculate cost devise new page replacement algorithm think optimal contorted test case belady anomaly occur new algorithm optimal explain answer segmentation similar paging use variable sized page define segment replacement algorithm base fifo page replacement scheme lru page replacement scheme remember segment size seg- ment choose replacement small leave consecutive location need segment consider strategy system segment relocate strategy system consider demand page computer system degree multi- programming currently fix system recently mea- sure determine utilization cpu paging disk alternative result show case happen degree multiprogramming increase increase cpu utilization paging help cpu utilization 13 percent disk utilization 97 percent cpu utilization 87 percent disk utilization 3 percent cpu utilization 13 percent disk utilization 3 percent operate system machine use base limit register modify machine provide page table page table set simulate base limit register enhanced clock algorithm discuss carr hennessy 1981 mcdougall mauro 2007 discuss virtual memory solaris virtual memory technique linux describe love 2010 mauerer 2008 freebsd describe mckusick et al 2015 carr hennessy 1981 w. r. carr j. l. hennessy wsclock sim- ple effective algorithm virtual memory management proceedings acm symposium operating systems principles 1981 page 87–95 p. j. denning working set model program behavior communications acm volume 11 number 5 1968 page 323–333 r. love linux kernel development edition developer w. mauerer professional linux kernel architecture john wiley sons 2008 mcdougall mauro 2007 r. mcdougall j. mauro solaris internals second edition prentice hall 2007

mckusick et al 2015 m. k. mckusick g. v. neville neil r. n. m. wat- son design implementation freebsd unix operating system second edition pearson 2015 russinovich et al 2017 m. russinovich d. a. solomon a. ionescu win- dows internals 1 seventh edition microsoft press 2017 chapter 10 exercise assume program reference address virtual mem- ory describe scenario following occur scenario occur explain tlb miss page fault tlb miss page fault tlb hit page fault tlb hit page fault asimplified view thread state ready run block thread ready wait schedule run processor block example wait o assume thread running state answer follow ques- tion explain answer thread change state incur page fault state change thread change state generate tlb miss resolve page table state change thread change state address reference resolve page table state change consider system use pure demand paging process start execution characterize page fault rate work set process load memory characterize page fault rate assume process change locality size new work set large store available free memory identify option system designer choose handle situation follow page table system 12 bit virtual physical address 256 byte page free page frame allocate order 9 f d.

dash page frame indicate page memory 0 x 4 0 x b 0 x 0 x 2 0 x c 0 x 0 0 x 1 convert following virtual address equivalent physical address hexadecimal number give hexadecimal case page fault use free frame update page table resolve logical address correspond physical copy write feature circumstance use beneficial hardware support require implement certain computer provide user virtual memory space 232 byte computer 222 byte physical memory virtual memory implement paging page size 4,096 byte user process generate virtual address 11123456 explain system establish corresponding physical location distinguish software hardware operation assume demand page memory page table hold register take 8 millisecond service page fault frame available replace page modify 20 millisecond replace page modify memory access time assume page replace modify 70 percent time maximum acceptable page fault rate effective access time 200 nanosecond consider page table system 16 bit virtual physical address 4,096 byte page reference bit page set 1 page ref- erence periodically thread zero value reference bit dash page frame indicate page memory page replacement algorithm localize lru number provide decimal convert following virtual address hexadecimal equivalent physical address provide answer hexadecimal decimal set reference bit appro- priate entry page table address guide provide example logical address hexadecimal result page fault set page frame lru page replacement algorithm choose resolve page fault page fault occur process request page block wait page bring disk physical mem- ory assume exist process user level thread mapping user thread kernel thread user thread incur page fault access stack user thread belong process affect page fault wait fault page bring memory explain apply 1 fifo 2 lru 3 optimal opt replacement algo- rithm following page reference string 2 6 9 2 4 2 1 7 3 0 5 2 1 2 9 5

1 7 3 0 5 2 1 2 9 5 7 3 8 5 0 6 3 0 2 6 3 5 2 4 1 3 0 6 1 4 2 3 5 7 3 1 4 2 5 4 1 3 5 2 0 1 1 0 2 3 4 5 0 1 4 2 1 7 9 8 3 5 2 6 8 1 0 7 2 4 1 3 5 8 0 1 2 3 4 4 3 2 1 0 0 1 2 3 4 4 3 2 1 0

indicate number page fault algorithm assume demand paging frame assume monitor rate pointer clock algorithm move pointer indicate candidate page replacement system notice pointer move fast pointer move slow discuss situation frequently lfu page- replacement algorithm generate few page fault recently lru page replacement algorithm discuss circumstance opposite hold discuss situation frequently mfu page- replacement algorithm generate few page fault recently lru page replacement algorithm discuss circumstance opposite hold khie pronounce k hi operate system use fifo replace- ment algorithm resident page free frame pool recently page assume free frame pool manage lru replacement policy answer following question page fault occur page exist free frame pool free space generate newly request page page fault occur page exist free frame pool resident page set free frame pool manage space request page system degenerate number resident page set system degenerate number page free frame pool zero consider demand page system following time measure o device follow indicate likely improve cpu utilization explain answer install fast cpu install big page disk increase degree multiprogramming decrease degree multiprogramming install main memory install fast hard disk multiple controller multiple add prepaging page fetch algorithm increase page size explain minor page fault time resolve major page explain compressed memory operate system suppose machine provide instruction access mem- ory location level indirect addressing scheme sequence page fault incur page program currently nonresident instruction program indirect memory load operation happen operate system process frame allocation technique page allocate process consider page reference page represent locality time x suppose replacement policy paged system examine page regularly discard page examination gain lose policy lru second chance replacement page replacement algorithm minimize number page fault achieve minimization distribute heavily page evenly memory have compete small number page frame associate page frame counter number page associate frame replace page search page frame counter number page associate frame replace page search page frame small define page replacement algorithm basic idea specif- ically address problem initial value counter counter increase counter decrease page replace select page fault occur algorithm follow reference string page frame 1 2 3 4 5 3 4 1 6 7 8 7 8 9 7 8 9 5 4 5 4 2

minimum number page fault optimal page- replacement strategy reference string b consider demand page system page disk average access transfer time 20 millisecond address translate page table main memory access time 1 microsecond memory access memory reference page table take access improve time add associative memory reduce access time memory reference page table entry associative memory

assume 80 percent access associative memory remain 10 percent 2 percent total cause page fault effective memory access time cause thrashing system detect thrashing detect thrashing system eliminate possible process work set represent datum represent code explain consider parameter δ define work set window work set model δ set low value effect page fault frequency number active nonsuspended process currently execute system effect δ set high value 1,024 kb segment memory allocate buddy system figure 10.26 guide draw tree illustrate following memory request allocate request 5 kb request 135 kb request 14 kb request 3 kb request 12 kb modify tree follow release memory perform coalesce possible release 3 kb release 5 kb release 14 kb release 12 kb system provide support user level kernel level thread mapping system correspond kernel thread user thread multithreaded process consist work set entire process b work set slab allocation algorithm use separate cache different object type assume cache object type explain scheme scale multiple cpu address scalability issue consider system allocate page different size process advantage paging scheme modifica- tion virtual memory system need provide write program implement fifo lru optimal opt page replacement algorithm present section 10.4 pro- gram initially generate random page reference string page number range 0 9 apply random page reference string algorithm record number page fault incur algorithm pass number page frame program startup implement program programming language choice find implementation fifo lru helpful virtual memory manager programming project design virtual memory manager project consist write program translate logical physical address virtual address space size 216 = 65,536 byte program read file contain logical address tlb page table translate logical address corresponding physical address output value byte store translate physical address learn goal use simulation understand step involve translate logical physical address include resolve page fault demand paging

translate logical physical address include resolve page fault demand paging manage tlb implement page replacement program read file contain 32 bit integer number represent logical address need concern 16- bit address mask rightmost 16 bit logical address 16 bit divide 1 8 bit page number 2 8 bit page offset address structure show specific include follow 28 entry page table page size 28 byte 16 entry tlb frame size 28 byte 256 frame physical memory 65,536 byte 256 frame × 256 byte frame size additionally program need concern read logical address translate correspond physical address need support write logical address space program translate logical physical address tlb page table outline section 9.3 page number extract logical address tlb consult case tlb hit frame number obtain tlb case tlb miss page table consult case frame number obtain page table page fault occur visual representation address- translation process handle page fault program implement demand paging describe section 10.2 back store represent file backing store.bin binary file size 65,536 byte page fault occur read 256 byte page file backing store store available page frame physical memory example logical address page number 15 result page fault program read page 15 backing store remem- ber page begin 0 256 byte size store page frame physical memory frame store page table tlb update subsequent access page 15 resolve tlb page table need treat backing store.bin random access file randomly seek certain position file read suggest standard c library function perform o include fopen fread fseek fclose size physical memory size virtual address space—65,536 byte need concern page replace- ment page fault later describe modification project small physical memory point page replacement strategy require provide file addresses.txt

contain integer value represent- e logical address range 0to65535 size virtual address space program open file read logical address translate corresponding physical address output value sign byte physical address begin write simple program extract page number offset base following integer number 1 256 32768 32769 128 65534 33153 easy way operator bit masking bit shifting correctly establish page number offset integer number ready begin initially suggest bypass tlb use page table integrate tlb page table work properly remember address translation work tlb tlb make fast ready implement tlb recall sixteen entry need use replacement strategy update tlb use fifo lru policy update tlb run program program run follow program read file addresses.txt contain 1,000 logical address range 0 65535 program translate logical address physical address determine content sign byte store correct physical address recall c language char data type occupy byte storage suggest char value program output following value 1 logical address translate integer value read 2 corresponding physical address program translate logical address 3 sign byte value store physical memory translate phys- provide file correct.txt contain correct output value file addresses.txt use file determine program correctly translate logical physical address completion program report following statistic 1 page fault rate percentage address reference result tlb hit rate percentage address reference resolve logical address addresses.txt

generate randomly reflect memory access locality expect high tlb hit far project assume physical memory size virtual address space practice physical memory typically small virtual address space phase project assume small physical address space 128 page frame 256 change require modify program keep track free page frame implement page replacement policy fifo lru section 10.4 resolve page fault free memory computer system provide mass storage permanently store file datum modern computer implement mass storage secondary storage hard disk nonvolatile memory device secondary storage device vary aspect transfer character time block character access sequentially randomly transfer datum syn- chronously asynchronously dedicate share read read write vary greatly speed way slow major component computer device variation operate system need provide wide range functionality application control aspect device key goal operate system o subsystem provide simple interface possible rest system device performance bottleneck key optimize o maximum concurrency c h p t e r chapter discuss mass storage nonvolatile storage sys- tem computer structure main mass storage system modern computer secondary storage usually provide hard disk drive hdd nonvolatile memory nvm device system slow large tertiary storage generally consist magnetic tape optical disk cloud storage common important storage device modern com- puter system hdd nvm device bulk chapter devote discuss type storage describe physical struc- ture consider scheduling algorithm schedule order o maximize performance discuss device formatting manage- ment boot block damage block swap space finally examine structure raid system type mass storage use general term non- volatile storage nvs talk storage drive discussion include type particular device hdd nvm device specify appropriate describe physical structure secondary storage device effect device structure use explain performance characteristic mass storage device evaluate o scheduling algorithm discuss operating system service provide mass storage include overview mass storage structure bulk secondary storage modern computer provide hard disk

storage structure bulk secondary storage modern computer provide hard disk drive hdds nonvolatile memory nvm device section hdd move head disk mechanism describe basic mechanism device explain operat- e system translate physical property logical storage address hard disk drives conceptually hdd relatively simple figure

11.1 disk platter flat circular shape like cd common platter diameter range 1.8 3.5 inch surface platter cover magnetic material store information record magnetically platter read information detect magnetic pattern platter read write head fly surface platter head attach disk arm move head unit surface platter logically divide circular track subdivide sector set track give arm position cylinder thousand concentric cylinder disk drive track contain hundred sector sector fix size small unit transfer sector size commonly 512 byte 2010 point manufacturer start migrate 4 kb sector storage capacity common disk drive measure gigabyte terabyte disk drive cover remove show figure 11.2 adisk drive motor spin high speed drive rotate 60 250 time second specify term rotation minute rpm common drive spin 5,400 7,200 10,000 15,000 rpm drive power use spin receive o request rotation speed relate transfer rate transfer rate rate datum flow drive computer performance aspect positioning time random access time consist part time necessary disk arm

desire cylinder call seek time time necessary overview mass storage structure 3.5 inch hdd cover remove desire sector rotate disk head call rotational latency typical disk transfer ten hundred megabyte datum second seek time rotational latency millisecond increase performance have dram buffer drive controller disk head fly extremely thin cushion measure micron air gas helium danger head contact disk surface disk platter coat thin protective layer head damage magnetic surface accident call head crash head crash normally repair entire disk replace datum disk lose back storage raid protect raid discuss section hdd seal unit chassi hold hdd allow removal shut system storage chassis helpful system need storage connect give time necessary replace bad drive work type storage medium removable include cd dvd blu ray disc disk transfer rate aspect computing publish performance number disk real world performance number state transfer rate high effective transfer rate example transfer rate rate bit read magnetic medium disk head different rate block deliver operate system nonvolatile memory devices nonvolatile memory nvm device grow importance simply describe nvm device electrical mechanical commonly device compose controller flash nand die semiconductor chip store datum nvm technology exist like dram battery backing lose content semiconductor technology like 3d xpoint far common discuss book overview nonvolatile memory devices flash memory base nvm frequently disk drive like container case call solid state disk ssd figure 11.3 instance take form usb drive know thumb drive flash drive dram stick surface mount motherboard main storage device like smartphone form act treat way discussion nvm device focus technology nvm device reliable hdd move part fast seek time rotational latency addition consume power negative expensive megabyte traditional hard disk capacity large hard disk time capacity nvm device increase fast hdd capacity price drop quickly use increase dramatically fact ssd similar device laptop computer small

use increase dramatically fact ssd similar device laptop computer small fast nvm device fast hard disk drive standard bus interface cause major limit throughput nvm device design connect directly system bus pcie example technology change traditional aspect computer design 3.5 inch ssd circuit board overview mass storage structure system use direct replacement disk drive use new cache tier move datum magnetic disk nvm main memory optimize performance nand semiconductor characteristic present storage reliability challenge example read write page increment similar sector datum overwrite nand cell erase erasure occur block increment page size take time read fast operation write slow read fast erase help situation nvm flash device compose die datapath die operation happen parallel datapath nand semiconductor deteriorate erase cycle approximately 100,000 program erase cycle specific number vary depend medium cell long retain datum write wear move part nand nvm lifespan measure year drive writes day dwpd measure time drive capacity write day drive fail example 1 tb nand drive 5 dwpd rating expect 5 tb day write warranty period limitation lead ameliorate algorithm fortunately usually implement nvm device controller con- cern operate system operate system simply read write logical block device manage logical block discuss detail section 11.1.5 nvm device perfor- mance variation base operate algorithm brief discussion controller warrant nand flash controller algorithm nand semiconductor overwritten write usually page contain invalid datum consider file system block write later write erase occur meantime page write old datum invalid second page current good version block nand block contain valid invalid page show figure 11.4 track logical block contain valid datum controller maintain flas translation layer ftl table map physical page contain currently valid logical block nand block valid invalid page track physical block state block contain invalid page erase consider ssd pende write request ssd page write block contain valid

ssd pende write request ssd page write block contain valid datum case write wait erase occur write occur free block space available individual page hold invalid datum case garbage collection occur good datum copy location free block erase receive write garbage collection store good datum solve problem improve write performance nvm device use over-provisioning device set aside number page frequently 20 percent total area available write block totally invalid garbage collection write operation invalidate old version datum erase place provision space device return free pool provision space help wear leveling block erase repeatedly frequently erase block wear fast entire device short lifespan block wear concurrently controller try avoid algorithm place datum erase block subsequent erase happen block erase block level wear entire device term datum protection like hdd nvm device provide error- correct code calculate store datum writing read datum detect error correct possible error correct code discuss section 11.5.1 page frequently correctible error page mark bad subsequent write generally single nvm device like hdd catastrophic failure corrupt fail reply read write request allow datum recoverable instance raid protection odd discuss volatile memory chapter mass storage structure justifiable dram frequently mass storage device specifically ram drive know name include ram disk act like secondary storage create device driver carve section system dram present rest system storage device drive raw block device commonly file system create standard file computer buffering caching purpose use dram temporary datum storage dram volatile datum ram drive survive system crash shutdown power cache buffer allocate programmer operate sys- tem ram drive allow user programmer place overview mass storage structure magnetic tape early secondary storage medium nonvolatile hold large quantity datum access time slow compare main memory drive addition random access magnetic tape thousand time slow random access hdd thousand time slow random access ssd tape

random access hdd thousand time slow random access ssd tape useful secondary storage tape mainly backup storage infrequently information medium transfer information system tape keep spool wind rewind past read write head move correct spot tape minute posi- tione tape drive read write datum speed comparable hdd tape capacity vary greatly depend particular kind tape drive current capacity exceed terabyte tape build compression double effective storage tape driver usually categorize width include 4 8 19 millimeter 1/4 1/2 inch name accord technology lto-6 figure 11.5 sdlt lto-6 tape drive tape cartridge insert datum memory temporary safekeeping standard file operation fact ram drive functionality useful drive find major operate system

linux /dev ram macos diskutil command create windows party tool solaris linux create /tmp boot time type tmpfs ram drive ram drive useful high speed temporary storage space nvm device fast dram fast o operation ram drive fast way create read write delete file content program use benefit ram drive store temporary file example program share datum easily write read file ram drive example linux boot time create temporary root file system initrd allow part system access root file system content part operate system understand storage device load secondary storage connection method secondary storage device attach computer system bus o bus kind bus available include advanced technology attachment ata serial ata sata esata serial attach scsi sas uni- versal serial bus usb fibr channel fc common connection method sata nvm device fast hdd industry create special fast interface nvm device call nvm express nvme nvme directly connect device system pci bus increase throughput decrease latency compare connection method data transfer bus carry special electronic processor call controller host bus adapter hba host controller controller computer end bus device controller build storage device perform mass storage o operation computer place command host controller typically memory map o port describe section 12.2.1 host controller send command message device controller controller operate drive hardware carry command device controller usually build cache data transfer drive happen cache storage medium data transfer host fast electronic speed occur cache host dram dma storage device address large dimensional array logical block logical block small unit transfer logical block map physical sector semiconductor page dimensional array logical block map sector page device sector 0 sector track outermost cylinder hdd example mapping proceed order track rest track cylinder rest cylinder outermost innermost nvm mapping tuple finite ordered list chip block page array logical block logical block address lba easy

block page array logical block logical block address lba easy algorithm use sector cylinder head tuple chip block page tuple mapping hdd theory convert logical block number old style disk address consist cylinder number track number cylinder sector number track practice difficult perform translation reason drive defective sector mapping hide substitute spare sector drive logical block address stay sequential physical sector location change second number sector track constant drive disk man- ufacturer manage lba physical address mapping internally current drive little relationship lba physical sector spite physical address vagary algorithm deal hdd tend assume logical address

relatively related physical address ascend logical address tend mean ascend physical address let look closely second reason medium use constant linear velocity clv density bit track uniform far track center disk great length sector hold outer zone inner zone number sector track decrease track outermost zone typically hold 40 percent sector track innermost zone drive increase rotation speed head move outer inner track rate datum move head method cd rom dvd rom drive alternatively disk rotation speed stay constant case density bit decrease inner track outer track data rate constant performance relatively matter data drive method hard disk know constant angular number sector track increase disk technology improve outer zone disk usually sector track similarly number cylinder disk increase large disk ten thousand cylinder note type storage device reasonable cover operating system text example shingled magnetic record hard drive high density bad perfor- combination device include nvm hdd technology volume manager section 11.5 knit nvm hdd device storage unit fast hdd low cost nvm device different characteristic common device need different caching scheduling algorithm maximize performance 11.2 hdd scheduling responsibility operate system use hardware efficiently hdd meet responsibility entail minimize access time maximize datum transfer bandwidth hdd mechanical storage device use platter access time major component mention section 11.1 seek time time device arm head cylinder contain desire sector rotational latency additional time platter rotate desire sector head device bandwidth total number byte transfer divide total time request service completion transfer improve access time bandwidth manage order storage o request process need o drive issue system operate system request specify piece information operation input output open file handle indicate file operate memory address transfer datum transfer desire drive controller available request service immediately drive controller busy new request service place queue pende request drive multiprogram- me system process device queue

queue pende request drive multiprogram- me system process device queue existence queue request device perfor- mance optimize avoid head seek allow device driver chance improve performance queue ordering past hdd interface require host specify track head use effort spend disk scheduling algorithm drive new turn century expose control host map lba physical address drive control current goal disk scheduling include fairness timeliness optimiza- tion bunching read write appear sequence drive perform well sequential o. scheduling effort

useful disk scheduling algorithm discuss note absolute knowledge head location phys- ical block cylinder location generally possible modern drive rough approximation algorithm assume increase lba mean increase physical address lbas close equate physical block simple form disk scheduling course come serve fcfs algorithm fifo algorithm intrinsically fair generally provide fast service consider example disk queue request o block cylinder 98 183 37 122 14 124 65 67 order disk head initially cylinder 53 53 98 183 37 122 14 124 65 finally 67 total head movement 640 cylinder schedule diagram figure 11.6 wild swing 122 14 124 illustrate problem schedule request cylinder 37 14 service request 122 124 total head movement decrease substantially performance improve scan algorithm disk arm start end disk move end service request reach cylinder get end disk end direction head movement reverse servicing continue head continuously scan forth disk scan algorithm call elevator algorithm disk arm behave like elevator building service request go reverse service request let return example illustrate apply scan schedule request cylinder 98 183 37 122 14 124 65 67 need know queue = 98 183 37 122 14 124 65 67 head start 53 fcfs disk scheduling direction head movement addition head current position assume disk arm move 0 initial head position 53 head service 37 14 cylinder 0 arm reverse end disk service request 65 67 98 122 124 183 figure 11.7 request arrive queue head service immediately request arrive head wait arm move end disk reverse direction come assume uniform distribution request cylinder consider density request head reach end reverse direction point relatively request immediately head cylinder recently service heavy density request end disk request wait long idea algorithm queue = 98 183 37 122 14 124 65 67

head start 53 scan disk scheduling circular scan c scan scheduling variant scan design provide uniform wait time like scan c scan move head end disk service request way head reach end immediately return beginning disk service request return trip let return example illustrate apply c scan sched- ule request cylinder 98 183 37 122 14 124 65 67 need know direction head movement request schedule assume request schedule disk arm move 0 199 initial head position 53 request serve depict figure 11.8 c scan scheduling algorithm essentially treat cylinder circular list wrap final cylinder selection disk scheduling algorithm disk scheduling algorithm include coverage rarely operating system designer decide implement deployer choose good use particular list request define optimal order retrieval computation need find optimal schedule justify saving scan scheduling algorithm performance depend heavily number type request instance suppose queue usually outstanding request scheduling algorithm behave choice disk head behave like fcfs scheduling scan c scan perform well system place heavy load disk likely cause starvation problem starvation drive linux create deadline scheduler scheduler maintain separate read write queue give read priority process likely block read write queue 53 65 67 queue = 98 183 37 122 14 124 65 67 head start 53 c scan disk scheduling sort lba order essentially implement c scan o request send batch lba order deadline keep queue read write sort lba fcfs check batch request fcfs queue old configure age default 500 m lba queue read write contain request select batch o. deadline o scheduler default linux redhat 7 distribu- tion rhel 7 include noop prefer cpu bind sys- tem fast storage nvm device completely fair queue- e scheduler cfq default sata drive cfq maintain queue insertion sort sort lba order real time good effort default idle exclusive priority order starvation possible use

good effort default idle exclusive priority order starvation possible use historical datum anticipate process likely issue o request soon determine idle wait new o ignore queue request minimize seek time assume locality reference storage o request process detail sched- uler find https://access.redhat.com/site/documentation/en-us /red hat enterprise linux/7 html performance tuning guide index.html 11.3 nvm scheduling disk scheduling algorithm discuss apply mechanical platter- base storage like hdd focus primarily minimize disk head movement nvm device contain move disk head commonly use simple fcfs policy example linux noop scheduler use fcfs policy modify merge adjacent request observed behavior nvm device indicate time require service read uniform property flash memory write service time uniform ssd scheduler exploit property merge adjacent write request service read request fcfs order see o occur sequentially randomly sequential access optimal mechanical device like hdd tape datum read write near read write head random access o measure input output operation second iops cause hdd disk head movement naturally random access o fast nvm hdd produce hundred iop ssd produce hundred thousand nvm device offer advantage raw sequential through- hdd head seek minimize reading writing datum medium emphasize case read performance type device range equivalent order magnitude advantage nvm device write nvm slow read decrease advantage furthermore write performance hdd consistent life device write performance nvm device vary depend device recall need garbage collection provisioning worn nvm device near end life erase cycle generally bad performance new way improve lifespan performance nvm device time file system inform device file delete device erase block file store approach discuss section 14.5.6 let look closely impact garbage collection performance consider nvm device random read write load assume block write free space available garbage collection occur reclaim space take invalid datum mean write cause read page write good datum page overprovisione space erase invalid data

page write good datum page overprovisione space erase invalid data block placement block overprovisione space summary write request eventually cause page write datum page read garbage collection page write good datum garbage collect block creation o request application call write amplificatio greatly impact write performance device bad case extra o trigger write 11.4 error detection correction error detection correction fundamental area computing include memory networking storage error detection determine problem occur example bit dram spontaneously change 0 1

content network packet change transmission block datum change write read detect issue system halt operation error propagate report error user administrator warn device start fail fail memory system long detect certain error parity bit scenario byte memory system parity bit associate record number bit byte set 1 parity = 0 odd parity = 1 bit byte damage 1 0 0 1 parity byte change match store parity similarly store parity bit damage match compute parity single bit error detect memory system double bit error undetected note parity easily calculate perform xor exclusive bit note byte memory need extra bit memory store parity parity form checksum use modular arithmetic compute store compare value fixed length word error detection method common networking cyclic redundancy check crcs use hash function detect multiple bit error error correction code ecc detect problem correct correction algorithm extra amount storage code vary base extra storage need error correct example disk drive use sector ecc storage device management flash drive page ecc controller write sector page datum normal o ecc write value calculate byte datum write sector page read ecc recalculate compare store value store calculate number different mismatch indicate datum corrupt storage medium bad section 11.5.3 ecc error correct contain information bit datum corrupt enable controller identify bit change calculate correct value report recoverable soft error change occur ecc correct error non- correctable hard error signal controller automatically ecc processing sector page read write error detection correction frequently differentiator con- sumer product enterprise product ecc system dram error correction data path protection example 11.5 storage device management operate system responsible aspect storage device management discuss drive initialization booting drive bad block recovery drive formatting partition volume new storage device blank slate platter magnetic recording material set uninitialized semiconductor storage cell storage device store datum divide sector controller read write nvm page

initialize ftl create process call low level formatting physical formatting low level formatting fill device special data structure storage location data structure sector page typically consist header data area trailer header trailer contain information controller sector page number error detection correction code drive low level format factory manu- facturing process formatting enable manufacturer test device initialize mapping logical block number defect free sector page medium usually possible choose sector size 512 byte 4 kb format disk large sector size mean few sector fit track mean few header trailer write track space available user datum operate system handle specific sector size use drive hold file operate system need record data structure device step step partition device group block page operate system treat partition separate device instance partition hold file system contain copy operate system executable code swap space file system contain user file operate system file system perform partitioning automatically entire device manage file system partition information write fixed format fix location storage device linux fdisk command manage partition storage device device recognize operate system partition information read operate system create device entry partition /dev linux configuration file /etc fstab tell operate system mount partition contain file system specified location use mount option read mount file system make file system available use system user second step volume creation management step implicit file system place directly partition volume ready mount time volume creation management explicit example multiple partition device raid set section 11.8 file system spread device linux volume manager lvm2 provide feature commercial party tool linux operate system zfs provide volume management file system integrate set command feature note volume mean mountable file system file contain file system step logical formatting creation file system step operate system store initial file system datum structure device data structure include map free allocate space

datum structure device data structure include map free allocate space initial directory partition information indicate partition contain bootable file system contain operate system partition label boot establish root file system mount device link device partition create generally computer file system consist mount volume windows separately name letter c d e system linux boot time boot file system mount file system mount tree structure discuss section 13.3 windows file system interface make clear give device linux single file access traverse device request file request file system volume access figure 11.9 show windows 7 disk management tool display volume c e f note e f partition disk 1 device unallocated space device partition possibly contain file system increase efficiency file system group block large chunk frequently call cluster device o block file system o cluster effectively assure o sequential access few random access characteristic file system try group file content near metadata reduce hdd head seek operate file operating system special program ability use partition large sequential array logical block file system datum structure array call raw disk o array term raw o. swap space section 11.6.2 example database system prefer raw o enable control storage device management windows 7 disk management tool show device partition volume file system exact location database record store raw o bypass file system service buffer cache file locking prefetching space allocation file name directory certain application efficient allow implement special purpose storage service raw partition application use provide file system manage datum note linux generally support raw o achieve similar access direct flag open system computer start run instance power reboot initial program run initial bootstrap loader tend simple computer bootstrap store nvm flash memory firmware system motherboard map know mem- ory location update product manufacturer need write virus infect system initialize aspect system cpu register

need write virus infect system initialize aspect system cpu register device controller content main tiny bootstrap loader program smart bring bootstrap program secondary storage bootstrap program store boot block fix location device default linux bootstrap loader grub2 https://www.gnu.org/software/grub/manual/ grub.html/ device boot partition call boot disk system code bootstrap nvm instruct storage controller read boot block memory device driver load point start execute code bootstrap program sophisticated bootstrap loader able load entire operate system non fixed location device start operate system run let consider example boot process windows note windows allow drive divide partition partition identify boot partition contain operate system device driver windows system place boot code logical block hard disk page nvm device term master boot booting storage device windows record mbr booting begin run code resident system firmware code direct system read boot code mbr understand storage controller storage device load sector addition contain boot code mbr contain table list partition drive flag indicate partition system boot illustrate figure 11.10 system identify boot partition read sector page partition call boot sector direct kernel continue remainder boot process include load subsystem system service disk move part small tolerance recall disk head fly disk surface prone failure failure complete case disk need replace content restore backup medium new disk frequently sector defective disk come factory bad block depend disk controller use block handle variety way old disk disk ide controller bad block handle manually strategy scan disk find bad block disk format bad block discover flag unusable file system allocate block bad normal operation special program linux badblock command run manually search bad block lock away datum reside bad block usually lose sophisticated disk smart bad block recovery con- troller maintain list bad block disk list initialize low level formatting factory update life disk low level formatting set aside spare

factory update life disk low level formatting set aside spare sector visible operate system controller tell replace bad sector logically spare sector scheme know sector sparing forwarding typical bad sector transaction follow operate system try read logical block 87 controller calculate ecc find sector bad report finding operate system o error device controller replace bad sector spare system request logical block 87 request translate replacement sector address controller note redirection controller invalidate opti- mization operate system disk scheduling algorithm reason disk format provide spare sector cylinder spare cylinder bad block remappe controller use spare sector cylinder possible alternative sector sparing controller instruct replace bad block sector slipping example suppose logical block 17 defective available spare follow sector 202 sector slipping remap sector 17 202 move spot sector 202 copy spare sector 201 202 200 201 sector 18 copy sector 19 slip sector way free space sector 18 sector 17 map recoverable soft error trigger device activity copy block datum block spare slip unrecoverable hard error result lose datum file block repair instance restoration backup tape require nvm device bit byte page nonfunc- tional manufacture time bad time management faulty area simple hdd seek time performance loss avoid multiple page set aside replacement location space provision area decrease usable capacity provision area way controller maintain table bad page set page available write access 11.6 swap space management swapping present section 9.5 discuss move entire process secondary storage main memory swapping set- ting occur physical memory reach critically low point process move memory swap space free available memory practice modern operate system implement swapping fashion system combine swapping virtual memory tech- nique chapter 10 swap page necessarily entire process fact system use term swapping paging interchangeably reflect merging concept swap space management low level task operate sys- tem virtual memory use secondary

low level task operate sys- tem virtual memory use secondary storage space extension main memory drive access slow memory access swap space significantly decrease system performance main goal design implementation swap space provide good throughput vir- tual memory system section discuss swap space swap space locate storage device swap space manage swap space way different operate system depend memory management algorithm use instance system implement swapping use swap space hold entire process image include code datum segment page system simply store page push main memory swap space need system vary megabyte disk space gigabyte depend physical memory virtual memory back way virtual memory note safe overestimate underestimate swap space require system run swap space force abort process crash entirely overestimation waste secondary storage space file harm system recommend set aside swap space solaris example suggest set swap space equal virtual memory exceed pageable physical memory past linux suggest set swap space double physical memory today paging algorithm change linux system use considerably swap space operate system include linux allow use multiple swap space include file dedicated swap partition swap space usually place separate storage device load place o system paging swapping spread system aswap space reside place carve normal file system separate partition swap space simply large file file system normal file system routine create allocate space alternatively swap space create separate raw partition file system directory structure place space separate swap space storage manager allocate deallocate block raw partition manager use algorithm optimize speed storage efficiency swap space access frequently file system recall swap space swapping paging internal fragmentation increase trade- acceptable life datum swap space generally short file file system swap space reinitialize boot time fragmentation short live raw partition approach create fix swap space disk partitioning add swap space require repartitione device involve move file system partition

swap space require repartitione device involve move file system partition destroy restore backup add swap space operate system flexible swap raw partition file system space linux example policy implementa- tion

separate allow machine administrator decide type swapping use trade convenience allocation management file system performance swap raw swap space management example illustrate swap space follow evolution swap- ping paging unix system traditional unix kernel start implementation swapping copy entire process contiguous disk region memory unix later evolve combination swapping paging paging hardware available solaris 1 sunos designer change standard unix method improve efficiency reflect technological development process

execute text segment page contain code bring file system access main memory throw away select pageout efficient reread page file system write swap space reread swap space back store page anonymous memory memory back file include memory allocate stack heap uninitialized datum change later version solaris big change solaris allocate swap space page force physical memory virtual memory page create scheme give well performance modern computer physical memory old system tend page linux similar solaris swap space anony- mous memory linux allow swap area establish swap area swap file regular file system dedicated swap partition swap area consist series 4 kb page slot hold swap page associate swap area swap map array integer counter correspond page slot swap area value counter 0 correspond page slot available value great 0 indicate page slot occupy swap page value counter indicate number mapping swap page example value 3 indicate swap page map different process occur swap page store region memory share process data structure swap linux system show figure 11.11 11.7 storage attachment computer access secondary storage way host attach storage network attach storage cloud storage swap file data structure swap linux system host attach storage storage access local o port port use technology common sata mention early typical system sata port allow system gain access storage individual storage device device chassis multiple drive chassis connect usb firewire thunderbolt port cable high end workstation server generally need storage need share storage use sophisticated o architecture fibr channel fc high speed serial architecture operate optical fiber conductor copper cable large address space switch nature communication multiple host storage device attach fabric allow great flexibility o communication wide variety storage device suitable use host attach storage hdd nvm device cd dvd blu ray tape drive storage area network sans discuss section 11.7.4 o command initiate

storage area network sans discuss section 11.7.4 o command initiate data transfer host attach storage device read write logical datum block direct specifically identify storage unit bus id target logical unit network attach storage nas figure 11.12 provide access storage network nas device special purpose storage system general computer system provide storage host network client access network attach storage remote procedure- interface nfs unix linux system cifs windows machine remote procedure call rpc carry tcp udp ip network usually local area network lan carry data traffic client network attach storage unit usually implement storage array software implement rpc interface cifs nfs provide locking feature allow sharing file host access nas protocol example user log multiple nas client access home directory client network attach storage provide convenient way computer lan share pool storage ease naming access enjoy local host attach storage tend efficient low performance direct attach storage option iscsi late network attach storage protocol essence use ip network protocol carry scsi protocol network scsi cable interconnect host storage result host treat storage directly attach storage distant host nfs cifs present file system send part file network iscsi send logical block network leave client use block directly create file system section 1.10.5 discuss cloud computing offering cloud provider cloud storage similar network attach storage cloud storage provide access storage network unlike nas storage access internet wan remote datum center provide storage fee free difference nas cloud storage storage access present user nas access file system cifs nfs protocol raw block device iscsi protocol operate system protocol integrate present nas storage way storage contrast cloud storage api base program use api access storage amazon s3 lead cloud storage offering dropbox example company provide app connect cloud storage provide example include microsoft onedrive apple icloud reason api instead exist protocol latency failure scenario

apple icloud reason api instead exist protocol latency failure scenario wan nas protocol design use lan low latency wan likely lose connectiv- ity storage user storage device lan connection fail system nfs cifs hang recover cloud storage failure like likely application simply pause access connectivity restore storage area network storage array drawback network attach storage system storage o operation consume bandwidth data network increase latency network communication problem particularly acute large client server installation communication server client compete bandwidth communication server astorage area network san private network storage protocol networking protocol connect server storage unit show figure 11.13 power san lie flexibility multiple host multiple storage array attach san storage dynamically allocate host storage array raid protect unprotected drive bunch disks jbod san switch allow prohibit access host storage example host run low disk space san configure allocate storage host san possible cluster server share storage storage array include multiple direct host connection sans typically port cost storage array san connectivity short distance typically routing nas connect host san storage array purpose build device figure 11.14 include san port network port contain drive store datum con- troller redundant set controller manage storage allow access storage network controller compose cpu memory software implement feature array include network protocol user interface raid protection snapshot repli- cation compression deduplication encryption function discuss chapter 14 storage array include ssd array contain ssd result- e maximum performance small capacity include mix ssd hdd array software administrator select good medium give use ssd cache hdd bulk storage array fc common san interconnect simplicity iscsi increase use san interconnect infiniban ib)—a special- purpose bus architecture provide hardware software support high speed interconnection network server storage unit 11.8 raid structure storage device continue small cheap eco- nomically feasible attach drive computer system have large number drive system present

attach drive computer system have large number drive system present opportunity improve rate datum read write drive operate paral- lel furthermore setup offer potential improve reliability datum storage redundant information store multiple drive failure drive lead loss datum variety disk organization technique collectively call redundant array inde- pendent disk raids commonly address performance past raid compose small cheap disk view cost- effective alternative large expensive disk today raid high reliability high data transfer rate economic rea- son raid stand inexpensive stand improvement reliability redundancy let consider reliability raid hdd chance disk set n disk fail great chance specific single disk fail suppose mean time failure mtbf single disk 100,000 hour mtbf disk array 100 raid storage structure variety way example system drive directly attach bus case operate system system software implement raid functionality alternatively intelligent host controller control multiple attach device implement raid device hardware finally storage array storage array discuss standalone unit controller cache drive attach host standard controller example fc common setup allow operate system software raid functionality raid protect storage disk 100,000/100 = 1,000 hour

41.66 day long store copy datum disk failure result loss significant datum high rate data loss unacceptable solution problem reliability introduce redundancy store extra information normally need event disk failure rebuild lose information disk fail datum lose raid apply nvm device nvm device move part likely fail hdd simple expensive approach introduce redundancy duplicate drive technique call mirroring mirroring logical disk consist physical drive write carry drive result call mirror volume drive volume fail datum read datum lose second drive fail fail drive replace mtbf mirror volume failure loss datum depend factor mtbf individual drive mean time repair time take average replace fail drive restore datum suppose failure drive independent failure connect failure mtbf single drive 100,000 hour mean time repair 10

failure mtbf single drive 100,000 hour mean time repair 10 hour mean time data loss mirror drive system 100 0002(2 10 = 500 106 hour 57,000 year aware assume drive failure independent power failure natural disaster earthquake fire flood result damage drive time manufacturing defect batch drive cause correlate failure drive age probability failure grow increase chance second drive fail repair spite considera- tion mirror drive system offer high reliability power failure particular source concern occur far frequently natural disaster mirroring drive write progress block drive power fail block fully write block inconsistent state solution problem write copy add solid state nonvolatile cache raid array write cache protect data loss power failure write consider complete point assume cache kind error protection correction ecc mirroring improvement performance parallelism let consider parallel access multiple drive improve perfor- mance mirroring rate read request handle double read request send drive long pair functional case transfer rate read single drive system number read unit time multiple drive improve transfer rate instead stripe datum drive simple form data striping consist split bit byte multiple drive striping call bit level striping example array drive write bit byte drive i. array drive treat single drive sector time normal size important time access rate drive participate access read write number access process second single drive access read time datum time single drive bit level striping generalize include number drive multiple 8 divide 8 example use array drive bit 4+i byte drive i. striping need occur bit level block level striping instance block file stripe multiple drive n drive block file go drive mod n)+1 level striping byte sector sector block possible block level striping commonly available striping parallelism storage system achieve striping main 1 increase throughput multiple small access page access load balancing 2 reduce response time large access

page access load balancing 2 reduce response time large access mirroring provide high reliability expensive striping provide high data transfer rate improve reliability numerous scheme provide redundancy low cost disk striping combine parity bit describe shortly propose scheme different cost performance trade off classify accord level call raid level describe common level figure 11.15 show pictorially figure p indicate error correct bit c indicate second copy datum case depict figure drive worth datum store extra drive store redundant information failure recovery raid level 0 raid level 0 refer drive array striping level block redundancy mirroring parity bit show figure 11.15(a raid level 1 raid level 1 refer drive mirroring figure 11.15(b show mirror organization raid level 4 raid level 4 know memory style error correcting- code ecc organization ecc raid 5 6 idea ecc directly storage array striping block drive example data block sequence write store drive 1 second block drive 2 nth block store drive n error correction calculation result block store drive n + 1 scheme show figure 11.15(c

drive label p store error correction block drive fail error correction code recalculation detect prevent datum pass request process throw raid 4 actually correct error ecc block take account fact unlike memory system drive controller detect sector read correctly single parity block error correction detection idea follow sector damage know exactly sector disregard datum sector use parity datum recalculate bad datum bit block determine 1 0 compute parity correspond bit sector drive parity remain bit equal store parity miss bit 0 1 block read access drive allow request process drive transfer rate large read high disk read parallel large write high transfer rate datum parity write parallel small independent write perform parallel operating- system write datum small block require block read modify new datum write parity block update know read modify write cycle single write require drive access read old block write new block wafl cover chapter 14 use raid level 4 raid level allow drive add raid set seamlessly add drive initialize block contain zero parity value change raid set correct raid level 4 advantage level 1 provide equal data protection storage overhead reduce parity drive need regular drive mirror drive need drive level 1 second read write series block spread multiple drive n way striping datum transfer rate read write set block n time fast level 1 performance problem raid 4 parity base raid level expense compute write xor parity overhead result slow write non parity raid array modern general purpose cpu fast compare drive o performance hit minimal raid storage array host bus adapter include hardware controller dedicated parity hardware controller offload parity computation cpu array array nvram cache store block parity compute buffer write controller drive buffering avoid read modify- write cycle gather datum write stripe write drive stripe concurrently combination hardware acceleration buffering parity raid

write drive stripe concurrently combination hardware acceleration buffering parity raid fast non- parity raid frequently outperform non caching non parity raid raid level 5 raid level 5 block interleave distribute parity differ level 4 spread datum parity n+1 drive store datum n drive parity drive set n block drive store parity store datum example array drive parity nth block store drive n mod 5 + 1 nth block drive store actual datum block setup show figure 11.15(d p distribute drive parity block store parity block drive drive failure result loss datum parity loss recoverable spread parity drive set raid 5 avoid potential overuse single parity drive occur raid 4 raid 5 common parity raid raid level 6 raid level 6 call p + q redundancy scheme like raid level 5 store extra redundant information guard multiple drive failure xor parity parity block identical provide recov- ery information instead parity error correct code galois fiel math calculate q. scheme show figure 11.15(e 2 block redundant datum store 4 block datum compare 1 parity block level 5 system tolerate drive multidimensional raid level 6 sophisticated storage array amplify raid level 6 consider array contain hundred drive put drive raid level 6 stripe result datum drive logical parity drive multidimensional raid level 6 logically arrange drive row column dimensional array implement raid level 6 horizontally row vertically column system recover failure multiple failure parity block location raid level show figure 11.15(f simplicity figure show raid parity dedicated drive reality raid block scatter row column raid level 0 + 1 1 + 0 raid level 0 + 1 refer combination raid level 0 1 raid 0 provide performance raid 1 provide reliability generally level provide well performance raid 5 common environment performance reliability important unfortunately like raid 1 double number drive need storage

important unfortunately like raid 1 double number drive need storage relatively expensive raid 0 + 1 set drive stripe stripe mirror equivalent raid variation raid level 1 + 0 drive mirror pair result mirror pair stripe scheme theoretical advantage raid 0 + 1 example single drive fail raid 0 + 1 entire stripe inaccessible leave stripe failure raid 1 + 0 single drive unavailable drive mirror available rest drive figure raid 0 1 1 single disk failure b raid 1 1 0 single disk failure raid 0 + 1 1 + 0 single disk failure numerous variation propose basic raid scheme describe result confusion exist exact definition different raid level implementation raid area variation consider follow layer raid implement volume management software implement raid kernel system software layer case storage hardware provide minimal feature raid solution raid implement host bus adapter hba hardware drive directly connect hba give raid set solution low cost flexible raid implement hardware storage array storage array create raid set level slice set small volume present operate system operate system need implement file system volume array multiple connection available san allow multiple host advantage array feature raid implement san interconnect layer drive virtualiza- tion device case device sit host storage accept command server manage access storage provide mirroring example write block separate storage device feature snapshot replication implement level snapshot view file system update take place snapshot cover fully chapter 14 repli- cation involve automatic duplication write separate site redundancy disaster recovery replication synchronous asyn- chronous synchronous replication block write locally remotely write consider complete asynchronous replication write group write periodically asyn- chronous replication result data loss primary site fail fast distance limitation increasingly replication data center host alternative raid protec- tion replication protect data loss increase read performance allow read replica copy course use storage

increase read performance allow read replica copy course use storage type raid implementation feature differ depend layer raid implement example raid implement software host need carry manage replication replication implement storage array san interconnect host operate system feature host datum replicate aspect raid implementation hot spare drive drive hot spare datum configure replacement case drive failure instance hot spare rebuild mirror pair drive pair fail way raid level reestablish automatically wait fail drive replace allocate hot spare allow failure repair human intervention select raid level give choice system designer choose raid level consideration rebuild performance drive fail time need rebuild datum significant important factor continuous supply datum require high performance interactive database system furthermore rebuild performance influence mean time failure rebuild performance vary raid level rebuilding easy raid level 1 datum copy drive level need access drive array rebuild datum fail drive rebuild time hour raid level 5 rebuild large drive raid level 0

high performance application data loss critical example scientific computing data set load explore raid level 0

work drive failure require repair reloading datum source raid level 1 popular application require high reliability fast recovery raid inserv storage array innovation effort provide well fast expensive solution frequently blur line separate previous technology consider inserv storage array hp 3par unlike storage array inserv require set drive configure specific raid level drive break 256 mb chunklet raid apply chunklet level drive participate multiple raid level chunklet multiple volume inserv provide snapshot similar create wafl file system format inserv snapshot read write read- allow multiple host mount copy give file system need copy entire file system change host make copy copy write reflect copy innovation utility storage file system expand shrink system original size size change require copy datum administrator configure inserv provide host large logical storage initially occupy small physical storage host start storage unused drive allocate host original logical level host believe large fixed storage space create file system drive add remove file system inserv file system notice change feature reduce number drive need host delay purchase drive need 0 + 1 1 + 0 performance reliability important example small database raid 1 high space overhead raid 5 prefer store moderate volume datum raid 6 multidimensional raid 6 common format storage array offer good performance protection large space overhead raid system designer administrator storage decision example drive give raid set bit protect parity bit drive array data transfer rate high system expensive bit protect parity bit space overhead parity bit low chance second drive fail fail drive repair great result datum loss concept raid generalize storage device include array tape broadcast datum wireless system apply array tape raid structure able recover datum tape array damage apply broadcast datum block datum split short unit broadcast parity unit unit receive reason reconstruct unit commonly tape drive robot contain multiple tape

reason reconstruct unit commonly tape drive robot contain multiple tape drive stripe datum drive increase throughput decrease backup time problem raid unfortunately raid assure datum available operate system user pointer file wrong example pointer file structure wrong incomplete write call tear write properly recover result corrupt datum process accidentally write file system structure raid protect physical medium error hardware software error failure hardware raid controller bug software raid code result total datum loss large landscape software hardware bug numerous potential peril datum solaris zfs file system take innovative approach solve problem use checksum zfs maintain internal checksum block include datum metadata checksum keep block checksumme store pointer block figure 11.17 consider inode data structure store file system metadata pointer datum inode checksum block datum problem datum checksum incorrect file system know datum mirror block correct checksum incorrect checksum zfs automatically update bad block good similarly directory entry point inode check- sum inode problem inode detect directory access checksumming take place zfs structure provide high level consistency error detection error cor- metadata block 1 metadata block 2 zf checksum metadata datum rection find raid drive set standard file system extra overhead create checksum calculation extra block read- modify write cycle noticeable overall performance zfs fast similar checksum feature find linux btrfs file system https://btrfs.wiki.kernel.org/index.php/btrfs design issue raid implementation lack flexibility con- sider storage array drive divide set drive set drive raid level 5 set result separate volume hold file system file system large fit drive raid level 5 set file system need little space factor know ahead time drive volume properly allocate frequently drive use requirement change time storage array allow entire set drive create large raid set issue arise volume size build set volume manager allow change volume size case leave issue describe mismatch file system size volume manager allow

leave issue describe mismatch file system size volume manager allow size change file system allow file system growth shrinkage volume change size file system need recreate advantage change zfs combine file system management volume management unit provide great functionality traditional separation function allow drive partition drive gather raid set pool storage pool hold zfs file system entire pool free space available file system pool zfs use memory model malloc free allocate release storage file system block free file system result artificial limit storage use need relocate file system volume resize volume zfs provide quota limit size file system reservation assure file system grow specified variable change file system owner time system like linux volume manager allow logical joining multiple disk create large disk volume hold large file system figure 11.18(a depict traditional volume file system figure 11.18(b show zfs model general purpose computer typically use file system store content user approach datum storage start storage pool place object pool approach differ file system way navigate pool find object user orient object storage computer orient design program typical sequence 1 create object storage pool receive object id 2 access object need object id 3 delete object object id traditional volume file system b zfs pooled storage traditional volume file system compare zfs model object storage management software hadoop fil hdfs ceph determine store object manage object protection typically occur commodity hardware raid array example hdfs store n copy object n different com- puter approach low cost storage array provide fast access object n system system hadoop cluster access object system copy fast access copy computation datum occur system result send network example system request system need network connectivity read write object there- fore object storage usually bulk storage high speed random access object storage advantage horizontal scalability storage array fix maximum

access object storage advantage horizontal scalability storage array fix maximum capacity add capacity object store simply add computer internal disk attach external disk add pool object storage pool petabyte key feature object storage object self describe include description content fact object storage know content addressable storage object retrieve base content set format content system store object storage common general purpose computer huge amount datum store object store include google internet search content dropbox content spotify song facebook photo cloud com- put

amazon aws generally use object store amazon s3 hold file system data object customer application run history object store http://www.theregister.co.uk/2016/07/15 /the history boys cas object storage map hard disk drive nonvolatile memory device major secondary storage o unit computer modern secondary storage struc- ture large dimensional array logical block drive type attach computer system way 1 local o port host computer 2 directly connect motherboard 3 communication network storage network connection request secondary storage o generate file system virtual memory system request specify address device reference form logical block number disk scheduling algorithm improve effective bandwidth hdd average response time variance response time algo- rithm scan c scan design improve- ment strategy disk queue ordering performance disk- scheduling algorithm vary greatly hard disk contrast solid state disk move part performance vary little scheduling algorithm simple fcfs strategy data storage transmission complex frequently result error error detection attempt spot problem alert system corrective action avoid error propagation error correction detect repair problem depend correction datum available datum corrupt storage device partition chunk space partition hold volume multidevice volume file system create volume operate system manage storage device block new device typically come pre formatted device partition file system create boot block allocate store system bootstrap pro- gram device contain operate system finally block page corrupt system way lock block replace logically spare efficient swap space key good performance system system dedicate raw partition swap space use file file system instead system allow user system administrator decision provide option storage require large system storage device fail way secondary storage device fre- quently redundant raid algorithm algorithm allow drive give operation allow continue operation automatic recovery face drive failure raid algorithm organize different level level provide combination reliability high transfer rate object storage big datum problem index inter- net cloud photo storage object

big datum problem index inter- net cloud photo storage object self define collection datum address object id file typically use replication data protection compute base datum system copy datum exist horizontally scalable vast capacity easy disk scheduling fcfs scheduling useful single user environment explain answer explain sstf scheduling tend favor middle cylinder innermost outermost cylinder rotational latency usually consider disk scheduling modify sstf scan c scan include latency important balance file system o disk controller system multitaske environment tradeoff involve reread code page file system versus swap space store way implement truly stable storage explain say tape sequential access medium hard disk random access medium fact suitability storage device random access depend transfer size term streaming transfer rate denote rate data transfer underway exclude effect access latency contrast effec- tive transfer rate ratio total byte total second include overhead time access latency suppose computer following characteristic level-2 cache access latency 8 nanosecond stream transfer rate 800 megabyte second main memory access latency 60 nanosecond stream transfer rate 80 megabyte second hard disk access latency 15 mil- lisecond stream transfer rate 5 megabyte second tape drive access latency 60 second stream transfer rate 2 megabyte second random access cause effective transfer rate device decrease datum transfer access time disk describe effective transfer rate average access follow stream transfer 1 512 byte 2 8 kilobyte 3 1 megabyte 4 16 megabyte utilization device ratio effective transfer rate stream transfer rate calculate utilization disk drive transfer size give a.

suppose utilization 25 percent high consider acceptable performance figure give compute small transfer size disk give acceptable utilization access device transfer large byte sequential access device small transfer compute minimum transfer size acceptable utiliza- tion cache memory tape tape random access device raid level 1 organization achieve well performance read request raid level 0 organization nonredundant striping datum reason use hdd secondary storage reason use nvm device secondary storage services 2012 provide overview datum storage variety modern computing environment discussion redundant array independent disk raids present patterson et al 1988 kim et al 2009 discuss disk scheduling algorithm ssd object base storage describe mesnier et al 2003 russinovich et al 2017 mcdougall mauro 2007 love 2010 discuss file system detail windows solaris linux respectively storage device continuously evolve goal increase perfor- mance increase capacity direction capacity improvement redhat linux distribution multiple selectable disk scheduling algorithm detail https://access.redhat.com/site/docume ntation en red hat enterprise linux/7 html performance tuning guide arelatively new file system btrfs detail https://btrfs.wiki.kernel.or history object store http://www.theregister.co.uk/2016/07/15 /the history boys cas object storage map kim et al 2009 j. kim y. oh e. kim j. c. d. lee s. noh disk sched- ulers solid state drivers proceedings seventh acm international confer- ence embedded software 2009 page 295–304 r. love linux kernel development edition developer mcdougall mauro 2007 r. mcdougall j. mauro solaris internals second edition prentice hall 2007 mesnier et al 2003 m. mesnier g. ganger e. ridel object base stor- age ieee communications magazine volume 41 number 8 2003 page 84–99 patterson et al 1988 d. a. patterson g. gibson r. h. katz case redundant arrays inexpensive disks raid proceedings acm sigmod international conference management data 1988 page 109

russinovich et al 2017 m. russinovich d. a. solomon a. ionescu win- dows internals 1 seventh edition microsoft press 2017 e. e. services information storage management store manage protect digital information classic virtualized cloud envi- ronment wiley 2012 chapter 11 exercise disk scheduling discipline fcfs truly fair star- vation occur explain assertion true describe way modify algorithm scan ensure explain fairness important goal multi user system example circumstance important operate system unfair serve o explain nvm device use fcfs disk scheduling algorithm suppose disk drive 5,000 cylinder number 0 4,999 drive currently serve request cylinder 2,150 previous request cylinder 1,805 queue pende request fifo 2,069 1,212 2,296 2,800 544 1,618 356 1,523 4,965 3,681 start current head position total distance cylinder disk arm move satisfy pende request follow disk scheduling algorithm elementary physics state object subject constant acceleration relationship distance d time t give d = 2at2 suppose seek disk exercise 11.14 accelerate disk arm constant rate half seek decelerate disk arm rate second half seek assume disk perform seek adjacent cylinder 1 millisecond stroke seek 5,000 cylinder 18 distance seek number cylinder head move explain seek time proportional square root seek distance write equation seek time function seek distance equation form t = x+y l t time millisecond l seek distance cylinder calculate total seek time schedule exercise 11.14

determine schedule fast small total seek time percentage speedup time save divide original time percentage speedup fast schedule suppose disk exercise 11.15 rotate 7,200 rpm average rotational latency disk drive seek distance cover time find compare contrast hdd nvm device good appli- cation type describe advantage disadvantage nvm device caching tier disk drive replacement compare compare performance c scan scan scheduling assum- ing uniform distribution request consider average response time time arrival request completion request service variation response time effective bandwidth performance depend relative size seek time rotational latency request usually uniformly distribute example expect cylinder contain file system metadata access frequently cylinder contain file suppose know 50 percent request small fixed number scheduling algorithm discuss chapter particularly good case explain answer propose disk scheduling algorithm give well per- formance take advantage hot spot disk consider raid level 5 organization comprise disk parity set block disk store fifth disk block access order perform follow write block datum write seven continuous block datum compare throughput achieve raid level 5 organization achieve raid level 1 organization follow read operation single block read operation multiple contiguous block compare performance write operation achieve raid level 5 organization achieve raid level 1 organization assume mixed configuration comprise disk orga- nize raid level 1 raid level 5 disk assume system flexibility decide disk organization use store particular file file store raid level 1 disk raid level 5 disk order optimize performance reliability storage device typically describe term mean time failure mtbf quantity call time mtbf actually measure drive hour failure system contain 1,000 disk drive 750,000- hour mtbf following well describe drive failure occur disk farm thousand year century decade year month week day hour minute second mortality statistic indicate average u.s. resident 1 chance 1,000 die age 20 21 deduce mtbf hour 20 year old die age 20 21 deduce mtbf hour 20 year old convert figure hour year mtbf tell expect lifetime 20 year old manufacturer guarantee 1 million hour mtbf certain model disk drive conclude number year drive warranty discuss relative advantage disadvantage sector sparing sector slipping discuss reason operate system require accurate information block store disk operat- e system improve file system performance knowledge write program implement following disk scheduling algo- program service disk 5,000 cylinder number 0 4,999 program generate random series 1,000 cylinder request service accord algorithm list program pass initial position disk head parameter command line report total head movement require algorithm c h p t e r main job computer o computing case main job o computing processing merely incidental instance browse web page edit file immediate interest read enter information compute answer role operate system computer o manage con- trol o operation o device related topic appear chapter bring piece paint complete picture o.

describe basic o hardware nature hardware interface place constraint internal facility operate system discuss o service provide operate system embodiment service application o interface explain operate system bridge gap hardware interface application interface discuss unix system v stream mechanism enable application assemble pipeline driver code dynamically finally discuss performance aspect o principle operating system design improve o performance explore structure operate system o subsystem discuss principle complexity o hardware explain performance aspect o hardware software control device connect computer major concern operating system designer o device vary widely func- tion speed consider mouse hard disk flash drive tape robot varied method need control method form o subsystem kernel separate rest kernel complexity manage o device o device technology exhibit conflict trend hand increase standardization software hardware interface trend help incorporate improved device generation existing computer operating system hand increasingly broad variety o device new device unlike previous device challenge incorporate computer operate system challenge meet combination hardware software technique basic o hardware element port bus device controller accommodate wide variety o device encapsulate detail oddity different device kernel operate system structure use device driver module device driver present uniform device- access interface o subsystem system call provide standard interface application operate system 12.2 o hardware computer operate great kind device fit general category storage device disk tape transmission device network con- nection bluetooth human interface device screen keyboard mouse audio device specialized involve steering jet aircraft human give input flight com- puter joystick foot pedal computer send output command cause motor rudder flap fuel engine despite incredible variety o device need concept understand device attach software control device communicate computer system send signal cable air device communicate machine connection point port example serial port term phy shorthand osi model physical layer reference port common data center nomenclature device share common

layer reference port common data center nomenclature device share common set wire connection call bus bus like pci bus computer today set wire rigidly define protocol specify set message send wire term electronic message convey pattern electrical voltage apply wire define timing device cable plug device b device b cable plug device c device c plug port computer arrangement call daisy chain daisy chain usually operate bus bus widely computer architecture vary signal- e method speed throughput connection method typical pc bus structure appear figure 12.1 figure pcie bus common pc system bus connect processor memory subsystem fast device expansion bus connect relatively slow device keyboard serial usb port low left portion figure disk connect serial attach scsi sas bus plug sas controller pcie flexible bus send datum lane lane compose signal pair pair receive datum transmit lane compose wire typical pc bus structure lane duplex byte stream transport data packet eight- bit byte format simultaneously direction physically pcie link contain 1 2 4 8 12 16 32 lane signify x prefix pcie card connector use 8 lane designate x8 example addition pcie go multiple generation come future example card pcie gen3 x8 mean work gen- eration 3 pcie use 8 lane device maximum throughput 8 gigabyte second detail pcie find https://pcisig.com

controller collection electronic operate port bus device serial port controller simple device controller single chip portion chip computer control signal wire serial port contrast fibr channel fc bus controller simple fc protocol complex data center pc fc bus controller implement separate circuit board host bus adapter hba)—that connect bus computer typically contain processor microcode private memory enable process fc protocol message device build controller look disk drive circuit board attach board disk controller implement disk protocol kind connection sas sata instance microcode processor task bad sector mapping prefetching buffering caching processor command datum controller accomplish o transfer short answer controller register datum control signal processor communicate controller read write bit pattern register way communication occur use special o instruction o address range hexadecimal serial port secondary serial port primary device o port location pc partial specify transfer byte word o port address o instruction trigger bus line select proper device bit device register alternatively device support memory map o. case device control register map address space processor cpu execute o request standard data- transfer instruction read write device control register map location physical memory past pc o instruction control device memory map o control figure 12.2 show usual o port address pc graphic controller o port basic control operation controller large memory map region hold screen content thread send output screen write datum memory map region controller generate screen image base content memory technique simple use write million byte graphic memory fast issue mil- lion o instruction time system move memory map o.

today o perform device controller o device control typically consist register call status control data data register data register read host input data register write host send output status register contain bit read host bit indicate state current command complete byte available read data register device error occur control register write host start command change mode device instance certain bit control register serial port choose duplex half duplex communication bit enable parity checking bit set word length 7 8 bit bit select speed support serial port datum register typically 1 4 byte size controller fifo chip hold byte input output datum expand capacity controller size datum register fifo chip hold small burst datum device host able receive datum complete protocol interaction host controller intricate basic handshaking notion simple explain hand- shaking example assume 2 bit coordinate producer consumer relationship controller host con- troller indicate state busy bit status register recall set bit mean write 1 bit clear bit mean write 0 controller set busy bit busy work clear busy bit ready accept command host signal wish command ready bit command register host set command ready bit command available controller execute example host write output port coordinate controller handshaking follow 1 host repeatedly read busy bit bit clear 2 host set write bit command register write byte data register 3 host set command ready bit 4 controller notice command ready bit set set 5 controller read command register see write command read data register byte o 6 controller clear command ready bit clear error bit status register indicate device o succeed clear busy bit indicate finish loop repeat byte step 1 host busy wait poll loop read status register busy bit clear controller device fast method reasonable wait long host probably switch task host know controller idle device host service

probably switch task host know controller idle device host service device quickly datum lose instance datum stream serial port keyboard small buffer controller overflow datum lose host wait long return read computer architecture cpu instruction cycle sufficient poll device read device register logical extract status bit branch zero clearly basic polling operation efficient polling inefficient attempt repeatedly rarely find device ready service useful cpu processing remain undone instance efficient arrange hardware controller notify cpu device ready service require cpu poll repeatedly o completion hardware mechanism enable device notify cpu call interrupt basic interrupt mechanism work follow cpu hardware wire call interrupt request line cpu sense execute instruction cpu detect controller assert signal interrupt request line cpu perform state save jump interrupt handler routine fix address memory interrupt han- dler determine cause interrupt perform necessary processing perform state restore execute return interrupt instruction return cpu execution state prior interrupt device controller raise interrupt assert signal interrupt request line cpu catch interrupt dispatch interrupt device driver initiate o cpu receive interrupt transfer control cpu execute check interrupt instruction return interrupt input ready output complete error generate interrupt signal interrupt drive o cycle latency command mac os x. handler handler clear interrupt service device figure 12.3 summarize interrupt drive o cycle stress interrupt management chapter single user modern system manage hundred interrupt second server hun- dred thousand second example figure 12.4 show latency command output macos reveal second quiet desktop computer perform 23,000 interrupt basic interrupt mechanism describe enable cpu respond asynchronous event device controller ready service modern operate system need sophisticated interrupt- 1 need ability defer interrupt handling critical processing 2 need efficient way dispatch proper interrupt handler device poll device raise 3 need multilevel interrupt operate system distin- guish high- low priority interrupt respond appropriate degree urgency multiple concurrent 4 need

priority interrupt respond appropriate degree urgency multiple concurrent 4 need way instruction operate system atten- tion directly separately o request activity page fault error division zero shall task accomplish trap modern computer hardware feature provide cpu interrupt controller hardware cpu interrupt request line nonmaskable interrupt reserve event unrecoverable memory error second interrupt line maskable turn cpu execution critical instruction sequence interrupt maskable interrupt device controller request service interrupt mechanism accept address number select specific interrupt handling routine small set architecture address offset table call interrupt vector vector contain memory address specialized interrupt handler purpose vectored interrupt mechanism reduce need single interrupt handler search possible source interrupt determine need service practice computer device interrupt handler address element interrupt vector common way solve problem use interrupt chaining element interrupt vector point head list interrupt handler interrupt raise handler corresponding list call find service request structure compromise overhead huge interrupt table inefficiency dispatch single interrupt handler figure 12.5 illustrate design interrupt vector intel pen- tium processor event 0 31 nonmaskable signal error condition cause system crash page fault need immediate action debugging request stop normal opera- tion jump debugger application event 32 255

maskable purpose device generate interrupt bind range exception device available coprocessor segment overrun reserved invalid task state segment segment present intel reserve use intel reserved use intel pentium processor event vector table interrupt mechanism implement system interrupt priority level level enable cpu defer handling low priority inter- rupt mask interrupt possible high priority interrupt preempt execution low priority interrupt modern operate system interact interrupt mechanism sev- eral way boot time operate system probe hardware bus determine device present install corresponding interrupt handler interrupt vector o device controller raise interrupt ready service interrupt signify output complete input datum available failure detect interrupt mechanism handle wide variety exception divide zero access protected nonexis- tent memory address attempt execute privileged instruction user mode event trigger interrupt common property occurrence induce operate system execute urgent self- interrupt handing case time resource constrain complicated implement system frequently split interrupt management leve interrupt handler flih second level interrupt handler slih flih perform context switch state storage queuing handling operation separately schedule slih perform handling request operation operate system good use interrupt exam- ple operate system use interrupt mechanism virtual memory paging page fault exception raise interrupt interrupt suspend current process jump page fault handler ker- nel handler save state process move process wait queue perform page cache management schedule o operation fetch page schedule process resume execution return example find implementation system call usually program use library call issue system call library routine check argument give application build data structure convey argu- ment kernel execute special instruction call software interrupt trap instruction operand identify desire kernel service process execute trap instruction interrupt hard- ware save state user code switch kernel mode dispatch kernel routine thread implement request service trap give relatively low interrupt priority compare assign device interrupt execute system behalf application urgent service device controller

device interrupt execute system behalf application urgent service device controller fifo queue overflow interrupt manage flow control ker- nel example consider case processing require complete disk read step copy datum kernel space user buffer copying time consume urgent block high- priority interrupt handling step start pende o disk drive step high priority disk effi- ciently need start o soon previous complete consequently pair interrupt handler implement kernel code complete disk read high priority handler record o status clear device interrupt start pende o raise low priority interrupt complete work later cpu occupy high- priority work low priority interrupt dispatch correspond handler complete user level o copy datum kernel buffer application space call scheduler place application thread kernel architecture suited implement multiple inter- rupt priority enforce precedence interrupt handling back- ground processing kernel application routine illustrate point solaris kernel solaris interrupt handler execute kernel thread range high scheduling priority reserve thread priority interrupt handler precedence application code kernel housekeeping implement priority relationship inter- rupt handler priority cause solaris thread scheduler preempt low- priority interrupt handler favor high priority one threaded implementation enable multiprocessor hardware run interrupt han- dler concurrently describe interrupt architecture linux chapter 20 windows10 chapter 21 unix appendix c. summary interrupt modern operate system handle asynchronous event trap supervisor mode routine kernel enable urgent work modern computer use system interrupt priority device controller hardware fault system call raise interrupt trigger kernel routine interrupt heavily time sensitive processing efficient interrupt handling require good system performance interrupt drive o common polling polling high throughput o. device driver use interrupt o rate low switch polling rate increase point polling fast efficient direct memory access device large transfer disk drive waste- ful use expensive general purpose processor watch status bit feed datum controller register byte time process term program o pio computer avoid

register byte time process term program o pio computer avoid burden main cpu pio offload work special purpose processor call direct- memory access dma controller initiate dma transfer host write dma command block memory block contain pointer source transfer pointer destination transfer count number byte transfer command block complex include list source destination address contiguous scatter gather method allow multiple transfer execute sin- gle dma command cpu write address command block dma controller go work dma controller proceed operate memory bus directly place address bus perform transfer help main cpu simple dma controller stan- dard component modern computer smartphone mainframe note straightforward target address kernel address space user space user example modify content space transfer lose set datum dma transfer datum user space thread access second copy operation time kernel memory user memory need double buffering inefficient time operate system move memory mapping section 12.2.1 perform o transfer directly device user address space handshake dma controller device controller perform pair wire call dma request dma acknowledge device controller place signal dma request wire word datum available transfer signal cause dma controller seize memory bus place desire address memory address wire place signal dma acknowledge wire device controller receive dma acknowledge signal transfer word datum memory remove dma request signal entire transfer finish dma controller interrupt cpu process depict figure 12.6 dma controller seize memory bus cpu momentarily prevent access main mem- ory access datum item cache cycle stealing slow cpu computation offload data transfer work dma controller generally improve total system performance computer architecture use physical memory address dma step dma transfer perform direct virtual memory access dvma virtual address undergo translation physical address dvma perform transfer memory map device intervention cpu use main memory protect mode kernel operate system generally prevent pro- cesse issue device command directly discipline protect

generally prevent pro- cesse issue device command directly discipline protect datum access control violation protect system erroneous use device controller cause system crash instead operat- e system export function sufficiently privileged process use access low level operation underlie hardware kernel memory protection process access device controller directly direct access achieve high performance avoid kernel com- munication context switch layer kernel software unfortunately interfere system security stability common general purpose oper- ating system protect memory device system try guard erroneous malicious application o hardware summary hardware aspect o complex consider level detail electronics hardware design concept describe sufficient enable understand o feature oper- ating system let review main concept bus controller o port register handshaking relationship host device controller execution handshaking polling loop interrupt offloading work dma controller large transfer give basic example handshaking take place device controller host early section reality wide variety available device pose problem operating system implementer kind device set capability control bit definition pro- tocol interact host different operate system design attach new device com- puter rewrite operate system device vary widely operate system convenient uniform o interface application address question 12.3 application o interface section discuss structuring technique interface oper- ating system enable o device treat standard uniform way explain instance application open file disk application o interface kernel o subsystem kernel o structure know kind disk new disk device add computer disruption operate system like complex software engineering problem approach involve abstraction encapsulation software layering specifically abstract away detailed difference o device identify gen- eral kind general kind access standardized set func- tion interface difference encapsulate kernel module call device driver internally custom tailor specific device export standard interface figure 12.7 illustrate o relate portion kernel structure software layer purpose device driver layer hide difference device controller o subsystem kernel o sys- tem call

difference device controller o subsystem kernel o sys- tem call encapsulate behavior device generic class hide hardware difference application make o subsystem indepen- dent hardware simplify job operating system developer benefit hardware manufacturer design new device compatible exist host controller interface sata write device driver interface new hardware popular operate sys- tem attach new peripheral computer wait operating system vendor develop support code unfortunately device hardware manufacturer type operate system standard device driver interface give device ship multiple device driver instance driver windows linux aix macos device vary dimension illustrate delay operation characteristic o device character stream block character stream device transfer byte block device transfer block byte unit sequential random access sequential device transfer datum fix order determine device user random access device instruct device seek available datum storage synchronous asynchronous synchronous device perform data transfer predictable response time coordination aspect system asynchronous device exhibit irregular unpredictable response time coordinate computer sharable dedicated sharable device concurrently process thread dedicated device speed operation device speed range byte second gigabyte second read write read write device perform input output support data transfer direction allow datum modify write write read purpose application access difference hide operate system device group conventional type result style device access find useful broadly applicable exact system call differ operate system device category fairly standard major access conven- application o interface tion include block o character stream o memory map file access network socket operate system provide special system call access additional device time day clock timer oper- ating system provide set system call graphical display video operate system escape door transpar- ently pass arbitrary command application device driver unix system ioctl o control ioctl system enable application access functionality implement device driver need invent new system ioctl system argument device identifier connect application driver refer hardware device

system argument device identifier connect application driver refer hardware device manage driver second integer select command implement driver pointer arbitrary data structure memory enable application driver communicate necessary control information datum device identifier unix linux tuple major minor device number major number device type second instance device example consider ssd device system issue command ls -l /dev sda following output brw rw---- 1 root disk 8 0 mar 16 09:18 /dev sda brw rw----

1 root disk 8 1 mar 16 09:18 /dev sda1 brw rw---- 1 root disk 8 2 mar 16 09:18 /dev sda2 brw rw---- 1 root disk 8 3 mar 16 09:18 /dev sda3 show 8 major device number operate system use information route o request appropriate device driver minor number 0 1 2 3

indicate instance device allow request o device entry select exact device request block character device block device interface capture aspect necessary access disk drive block orient device device expect understand command read write random access device expect seek command specify block trans- fer application normally access device file system interface read write seek capture essential behavior block storage device application insulate low level difference device operate system special application database management system prefer access block device simple linear array block mode access call raw o. application perform buffering file system cause extra unneeded buffering likewise application provide locking block region operate system lock service redundant contradictory bad avoid conflict raw device access pass control device directly application let operate system step way unfortunately operating system service perform device compromise common operate system allow mode operation file disable buffering locking unix world call direct o. memory map file access layer block device driver offer read write operation memory map interface provide access disk storage array byte main memory system map file memory return virtual memory address contain copy file actual data transfer perform need satisfy access memory image transfer handle mechanism demand page virtual memory access memory map o efficient memory mapping convenient programmer access memory map file simple read write memory operate system offer virtual memory commonly use mapping interface kernel service instance execute program operate system map executable memory transfer control entry address executable mapping interface commonly kernel access swap space disk keyboard example device access character- stream interface basic system call interface enable application character interface library build offer line time access buffering editing service example user type backspace precede character remove input stream style access convenient input device keyboard mouse modem produce datum input spontaneously time necessarily predict application access style good output device printer audio board

predict application access style good output device printer audio board naturally fit concept linear stream byte performance addressing characteristic network o differ significantly disk o operate system provide network o interface different read()–write()–seek interface disk interface available operating system include unix windows network socket interface think wall socket electricity electrical appliance plug analogy system call socket interface enable application create socket connect local socket remote address plug application socket create application listen remote application plug local socket send receive packet connection support implementation network server socket interface provide function call select manage set socket select return information socket packet wait receive socket room accept packet send use select eliminate polling busy waiting necessary network o. function encapsulate essential behavior network greatly facilitate application o interface creation distribute application use underlie network hardware protocol stack approach interprocess communication network com- munication implement instance windows provide interface network interface card second interface network protocol unix long history proving ground network technology find half duplex pipe duplex fifo duplex stream message queue socket information unix networking give clocks timers computer hardware clock timer provide basic current time elapsed time set timer trigger operation x time t. function heavily operate system time-sensitive application unfortunately system call implement function standardize operate system hardware measure elapsed time trigger operation call programmable interval timer set wait certain time generate interrupt set repeat process generate periodic interrupt scheduler use mechanism generate interrupt preempt process end time slice disk o subsystem use invoke periodic flushing dirty cache buffer disk network subsystem use cancel operation proceed slowly network congestion failure operate system provide interface user process use timer operate system support timer request number timer hardware channel simulate virtual clock kernel timer device driver maintain list interrupt want routine user request sort early time order set timer early time

user request sort early time order set timer early time timer interrupt kernel signal requester reload timer early time computer clock hardware variety purpose modern pc include high performance event timer hpet run rate 10 megahertz range comparator set trigger repeatedly value hold match hpet trigger generate interrupt operate system clock management routine determine timer action precision trigger limit resolution timer overhead maintain virtual clock furthermore timer tick maintain system time day clock system clock drift drift correct protocol design purpose ntp network time protocol use sophisticated latency calculation computer clock accurate atomic clock level computer hardware clock construct high frequency counter computer value counter read device register case counter consider high resolution clock clock generate interrupt offer accurate measurement time interval nonblocking asynchronous o aspect system interface relate choice block- e o nonblocking o. application issue block system execution call thread suspend thread move operate system run queue wait queue system com- plete thread move run queue eligible resume execution resume execution receive value return system physical action perform o device generally asynchronous varying unpredictable time nev- ertheless operate system provide block system call application interface block application code easy write nonblocke user level process need nonblocking o.

example user interface receive keyboard mouse input process dis- play datum screen example video application read frame file disk simultaneously decompress display output display way application writer overlap execution o write multithreaded application thread perform block system call continue execute operate system provide nonblocking o system call anonblocke halt execution thread extended time instead return quickly return value indicate byte transfer alternative nonblocking system asynchronous system asynchronous return immediately wait o complete thread continue execute code completion o future time communicate thread setting variable address space thread triggering signal software interrupt routine execute outside linear control flow thread difference nonblocking asynchronous system call nonblocking read return immediately datum available number byte request few asynchronous read request transfer perform entirety complete future time o method show figure 12.9 asynchronous activity occur modern operate system fre- quently expose user application contain operate system operation secondary storage device network o useful example default application issue network send request storage device write request operate system note request buffer o return application possible optimize overall system performance operate system complete request system failure occur interim application lose flight request operate system usually limit application o interface o method synchronous b asynchronous long buffer request version unix flush secondary storage buffer 30 second example request flush 30 second occurrence system provide way allow application request flush buffer like secondary storage buffer datum force secondary storage wait buffer flush interval data consistency application maintain kernel read datum buffer issue o request device ensure datum write return request reader note mul- tiple thread perform o file receive consistent datum depend kernel implement o. situation thread need use lock protocol o request need perform immediately o system call usually way indicate give request o specific device perform synchronously good example nonblocking behavior select system network socket system take argument

example nonblocking behavior select system network socket system take argument specify maxi- mum waiting time set 0 thread poll network activity block select introduce extra overhead select check o possible data transfer select follow kind read write command variation approach find mach block multiple read specify desire read device system return soon complete operate system provide major variation o appli- cation interface vectored o allow system perform multiple o operation involve multiple location example unix readv sys- tem accept vector multiple buffer read source vector write vector destination transfer cause individual invocation system call scatter gather method useful variety reason multiple separate buffer content transfer sys- tem avoid context switching system overhead vec- tore o datum need transfer large buffer right order transmit inefficient addition version scatter gather provide atomicity assure o interruption avoid corruption datum thread perform o involve buffer possible programmer use scatter gather o feature increase throughput decrease system 12.4 kernel o subsystem kernel provide service relate o. service scheduling buffering caching spooling device reservation error handling pro- vide kernel o subsystem build hardware device- driver infrastructure o subsystem responsible protect errant process malicious user schedule set o request mean determine good order execute order application issue system call rarely good choice scheduling improve overall system performance share device access fairly process reduce average waiting time o complete simple example illustrate suppose disk arm near beginning disk application issue block read call disk application 1 request block near end disk application 2 request near beginning application 3 request middle disk operate system reduce distance disk arm travel serve application order 2 3 1 rearrange order service way essence o scheduling operating system developer implement scheduling maintain wait queue request device application issue block o system request place queue device o scheduler rearrange order queue improve overall system

queue device o scheduler rearrange order queue improve overall system effi- ciency average response time experience application operat- e system try fair application receive especially poor service priority service delay sensitive request instance request virtual memory subsystem priority application request scheduling algorithm disk o detail section 11.2 kernel support asynchronous o able track o request time purpose operate system attach wait queue device status table kernel manage table contain entry o device show figure 12.10 table entry indicate device type address state function kernel o subsystem device laser printer device disk unit 1 device disk unit 2 disk unit 2 disk unit 2 idle busy device busy request type request parameter store table entry device schedule o operation way o subsystem improve efficiency computer way storage space main memory storage hierarchy buffering caching abuffer course memory area store datum transfer device device application buffering reason reason cope speed mismatch producer consumer datum stream suppose example file receive internet storage ssd network speed thousand time slow drive buffer create main memory accumulate byte receive network entire buffer datum arrive buffer write drive single operation drive write instantaneous network interface need place store additional incoming datum buffer network fill buffer drive write request network start fill second buffer buffer write storage time network fill second buffer drive write complete network switch buffer drive write second double buffering decouple producer datum consumer relax timing requirement need decoupling illustrate figure 12.11 list enormous difference device speed typical computer hardware interface second use buffering provide adaptation device different data transfer size disparity especially common computer networking buffer widely fragmentation common pc data center o device interface speed reassembly message send large message fragment small network packet packet send network receive place reassembly buffer form image source use buffering support copy semantic application o. example clarify source use buffering support copy semantic application o. example clarify meaning copy semantic suppose application buffer datum wish write disk call write system provide pointer buffer integer specify number byte write system return happen application change content buffer copy semantic version datum write disk guarantee version time application system independent subsequent change application buffer simple way operate system guarantee copy semantic write system copy application datum kernel buffer return control application disk write perform kernel buffer subsequent change application buffer effect copying datum kernel buffer application datum space common operate system despite overhead operation introduce clean semantic effect obtain efficiently clever use virtual memory mapping copy write page protection cache region fast memory hold copy datum access cache copy efficient access original instance instruction currently run process store disk cache physical memory copy cpu secondary primary cache kernel o subsystem difference buffer cache buffer hold exist copy datum item cache definition hold copy fast storage item reside caching buffering distinct function region memory purpose instance preserve copy seman- tic enable efficient scheduling disk o operate system use buffer main memory hold disk datum buffer cache improve o efficiency file share application write reread rapidly kernel receive file o request kernel access buffer cache region file available main memory physical disk o avoid defer disk write accumulate buffer cache second large transfer gather allow efficient write schedule strategy delay write improve o efficiency discuss context remote file access section 19.8 spooling device reservation spool buffer hold output device printer accept interleaved data stream printer serve job time application wish print output concurrently have output mix operate system solve problem intercept output printer application output spool separate secondary storage file application finish print spool system queue corresponding spool file output printer spool system copy queue spool file printer time operate system spooling

system copy queue spool file printer time operate system spooling manage system daemon process handle kernel thread case operate system provide control interface enable user system administrator display queue remove unwanted job job print suspend print printer service device tape drive printer usefully multiplex o request multiple concurrent application spooling way operating system coordinate concurrent output way deal concurrent device access provide explicit facility coordination operate system include vms provide support exclusive device access enable process allocate idle device deallocate device long need operate system enforce limit open file handle device operate system provide function enable process coordinate exclusive access instance windows provide system call wait device object available parameter openfile system declare type access permit concurrent thread system application avoid deadlock operate system use protect memory guard kind hardware application error complete system failure usual result minor mechanical malfunction device o transfer fail way transient reason network overloaded permanent reason disk controller defective operate system compensate effectively transient failure instance disk read failure result read retry network send error result resend protocol specify unfortunately important component experience permanent failure operate system unlikely recover general rule o system return bit information status signify success failure unix operate system additional integer variable name errno return error code value indicate general nature failure example argument range bad pointer file open contrast hardware provide highly detailed error infor- mation current operate system design convey information application instance failure scsi device report scsi protocol level detail sense key iden- tifie

general nature failure hardware error illegal request additional sense code state category failure bad command parameter self test failure additional sense code qualifie give detail command parameter error hardware subsystem fail self test scsi device maintain internal page error log information request host seldom error closely relate issue protection user process acci- dentally purposely attempt disrupt normal operation system attempt issue illegal o instruction use mechanism ensure disruption place system prevent user perform illegal o define o instruction privileged instruction user issue o instruction directly operate system o user program execute system request operate system perform o behalf figure 12.12 operate system execute monitor mode check request valid o request operate system return user addition memory map o port memory location protect user access memory protection system note kernel simply deny user access graphic game video editing playback software need direct access memory map graphic controller memory speed performance graphic example kernel case provide lock mechanism allow section graphic memory represent window screen allocate process time kernel data structure kernel need state information use o component variety kernel datum structure open file table kernel o subsystem use system perform o. structure discuss section 14.1 kernel use similar structure track network connection character device communication o unix provide file system access variety entity user file raw device address space process entity support read operation semantic differ instance read user file kernel need probe buffer cache decide perform disk o. read raw disk kernel need ensure request size multiple disk sector size align sector boundary read process image merely necessary copy datum memory unix encapsulate difference uniform structure object orient technique open file record show figure 12.13 contain dispatch table hold pointer appropriate routine depend type file operate system use object orient method exten- sively instance windows use message passing implementation o.

method exten- sively instance windows use message passing implementation o. o request convert message send kernel o manager device driver change message content output message contain datum write input message contain buffer receive datum message passing approach add overhead comparison procedural technique use share datum structure simplify structure design o system add flexibility system wide open file table pointer read write function pointer select function pointer ioctl function pointer close function networking socket record pointer network info pointer read write function pointer select function pointer ioctl function pointer close function unix o kernel structure computer reside datum center far remove issue power use power cost increase world increasingly troubled long term effect greenhouse gas emission datum cen- ter cause concern target increase efficiency electricity use generate heat computer component fail high temperature cooling equation consider cool- e modern data center use twice electricity power equipment approach data center power optimization use range interchange data center air air chill natural source lake water solar panel operate system play role power use heat gener- ation cooling cloud computing environment processing load adjust monitoring management tool evacuate user pro- cesse system idle system power load require use operate system analyze load suf- ficiently low hardware enable power component cpusand external o device cpu core suspend system load require resume load increase core need run queue thread state course need save suspend restore resume feature need server data center kernel o subsystem server use vast amount electricity disable unneeded core decrease electricity cooling need mobile computing power management high priority aspect operate system minimize power use maximize bat- tery life increase usability device help compete alternative device today mobile device offer functionality yesterday high- end desktop power battery small fit pocket order provide satisfactory battery life modern mobile operate system design ground power management

battery life modern mobile operate system design ground power management key feature let examine detail major feature enable popular android mobile system maximize battery life power collapse component- level power management wakelocks power collapse ability device deep sleep state device use marginally power fully power able respond external stimulus user press button time quickly power power collapse achieve power individual component device screen speaker o subsystem consume power operate system place cpu low sleep state modern arm cpu consume hundred milliwatt core typical load handful milliwatt low sleep state state cpu idle receive interrupt wake resume previous activity quickly idle android phone pocket use little power spring life receive phone android able turn individual component phone know safe power flash storage know power overall o subsystem answer component level power management infrastructure understand relationship component compo- nent use understand relationship component android build device tree represent phone physical device topology example topology flash usb storage sub node o subsystem sub node system bus turn connect cpu understand usage component associate device driver driver track component use example o pende flash application open reference audio subsystem information android manage power phone individual component component unused turn component system bus unused system bus turn component entire device tree unused system enter power collapse technology android aggressively manage power con- sumption final piece solution miss ability application temporarily prevent system enter power collapse consider user play game watch video wait web page open case application need way device awake temporarily wakelock enable functionality application acquire release wakelock need application hold wakelock kernel prevent system enter power collapse example android market update application hold wakelock ensure system sleep update complete complete android market release wakelock allow system enter power collapse power management general base device management complicated far portray boot time firmware system analyze

device management complicated far portray boot time firmware system analyze system hardware create device tree ram ker- nel use device tree load device driver manage device additional activity pertain device manage includ- ing addition subtraction device run system hot plug understand change device state power management modern general purpose computer use set firmware code advanced con- figuratio power interface acpi manage aspect hardware acpi industry standard http://www.acpi.info feature pro- vide code run routine callable kernel device state discovery management device error management power management example kernel need quiesce device call device driver call acpi routine talk device kernel o subsystem summary summary o subsystem coordinate extensive collection service available application part kernel o subsystem supervise procedure management space file device access control file device operation control example modem seek file system space allocation device allocation buffering caching spooling o scheduling device status monitoring error handling failure recovery device driver configuration initialization power management o device upper level o subsystem access device uniform interface provide device driver 12.5 transform o request hardware early describe handshaking device driver device controller explain operate system connect appli- cation request set network wire specific disk sector consider transform o request hardware operations example read file disk application refer datum file disk file system map file file system directory obtain space allocation file instance ms dos fat relatively simple operating file system today common interchange format map number indicate entry file access table table entry tell disk block allocate file unix map inode number correspond inode contain space allocation information connection file disk controller hardware port address memory map controller register method ms dos fat mention ms dos file precede colon string identify specific hardware device example c file primary hard disk fact c represent primary hard disk build operate system c map specific port address

hard disk build operate system c map specific port address device table colon separator device space separate file system space separation make easy operate system associate extra functionality device instance easy invoke spooling file write printer instead device space incorporate regular file system space unix normal file system service provide automatically file system provide ownership access control file name device owner access control file store device interface provide access o system level name access device access file store device unix represent device name regular file system space unlike ms dos fat file colon separator unix path clear separation device portion fact path device unix mount table associate prefix path name specific device name resolve path unix look mount table find long match prefix correspond entry mount table give device device form file system space unix look file system directory structure find inode number < major minor > device number major device number identify device driver call handle o device minor device number pass device driver index device table corresponding device table entry give port address memory map address device controller modern operate system gain significant flexibility multiple stage lookup table path request physical device con- troller mechanism pass request application driver general introduce new device driver com- puter recompile kernel fact operate system ability load device driver demand boot time system probe hardware bus determine device present load necessary driver immediately require o request device add boot detect error cause interrupt generate associate interrupt handler example prompt kernel inspect device detail load appropri- eat device driver dynamically course dynamic loading unloading complicated static loading require complex kernel algo- rithm device structure locking error handling forth describe typical life cycle block read request depict figure 12.14 figure suggest o operation require great step consume tremendous number cpu cycle 1 process issue

great step consume tremendous number cpu cycle 1 process issue block read system file descriptor file open previously 2 system code kernel check parameter correctness case input datum available buffer cache datum return process o request complete 3 physical o perform process remove run queue place wait queue device o request schedule eventually o subsystem send request device driver depend operate system request send subroutine kernel message life cycle o request 4 device driver allocate kernel buffer space receive datum schedule o. eventually driver send command device controller write device control register 5 device controller operate device hardware perform data 6 driver poll status datum set dma transfer kernel memory assume transfer manage dma controller generate interrupt transfer 7 correct interrupt handler receive interrupt interrupt- vector table store necessary datum signal device driver return interrupt 8

device driver receive signal determine o request complete determine request status signal kernel o subsystem request complete 9 kernel transfer datum return code address space request process move process wait queue ready queue 10 move process ready queue unblock process scheduler assign process cpu process resume execution completion system unix system v subsequent unix release interesting mech- anism call streams enable application assemble pipeline driver code dynamically astream duplex connection device driver user level process consist stream head interface user process driver end control device zero stream module stream head driver end compo- nent contain pair queue read queue write queue message passing transfer datum queue streams structure show figure 12.15 module provide functionality streams processing push stream use ioctl system example process open usb device like keyboard stream push module handle input editing message exchange queue adjacent module queue module overflow adjacent queue prevent occur queue support flo control flow control queue accept message immediately send queue adjacent module buffer queue support flow control buffer message accept message sufficient buffer space process involve exchange control message queue adjacent module streams structure user process write datum device write putmsg system write system write raw datum stream putmsg allow user process specify message regardless system user process stream head copy datum message deliver queue module line copying message continue message copy driver end device similarly user process read datum stream head read getmsg system read stream head get message adjacent queue return ordinary datum unstructured byte stream process getmsg message return process streams o asynchronous nonblocking user pro- cess communicate stream head write stream user process block assume queue use flow control room copy message likewise user process block read stream datum available mention driver end like stream head module read write queue driver end respond interrupt trigger frame ready read

write queue driver end respond interrupt trigger frame ready read network unlike stream head block unable copy message queue line driver end handle incoming datum driver support flow control device buffer device typically resort drop incoming message consider network card input buffer network card simply drop message buffer space store incoming message benefit streams provide framework modular incremental approach write device driver network protocol module different stream different device example network module ethernet network card 802.11 wireless network card furthermore treat character device o unstructured byte stream streams allow sup- port message boundary control information communicate module unix variant support streams prefer method write protocol device driver example system v unix solaris implement socket mechanism streams o

major factor system performance place heavy demand cpu execute device driver code schedule process fairly efficiently block unblock result context switch stress cpu hardware cache o expose inefficiency interrupt handling mechanism kernel addition o load memory bus datum copy controller physical memory copy kernel buffer application datum space cope gracefully demand major concern computer modern computer handle thousand interrupt second interrupt handling relatively expensive task interrupt cause system perform state change execute interrupt handler restore state program o efficient interrupt- drive o number cycle spend busy waiting excessive o completion typically unblock process lead overhead network traffic cause high context switch rate consider instance remote login machine character type local machine transport remote machine local machine character type keyboard interrupt generate character pass interrupt handler device driver kernel user process user process issue network o system send character remote machine character flow local kernel network layer construct network packet network device driver network device driver transfer packet network controller send character generate interrupt interrupt pass kernel cause network o system complete remote system network hardware receive packet interrupt generate character unpack network proto- cols give appropriate network daemon network daemon identify remote login session involve pass packet appropriate subdaemon session flow con- text switch state switch figure 12.16 usually receiver echo character sender approach double work system use separate end processor terminal o reduce interrupt burden main cpu instance terminal concentrator multiplex traffic hundred remote terminal port large computer o channel dedicated special purpose cpu find mainframe high end system job channel offload o work main cpu idea channel datum flow smoothly main cpu remain free process datum like device controller dma controller find small computer channel process general sophisticated program channel tune particular workload employ principle improve efficiency o reduce number context switch reduce number time datum copy memory

reduce number context switch reduce number time datum copy memory pass device application reduce frequency interrupt large transfer smart con- troller polling busy waiting minimize increase concurrency dma knowledgeable controller chan- nel offload simple datum copying cpu processing primitive hardware allow operation device controller concurrent cpu bus operation balance cpu memory subsystem bus o performance overload area cause idleness o device vary greatly complexity instance mouse simple mouse movement button click convert numeric value pass hardware mouse device driver application contrast functionality provide windows disk device driver complex manage individual disk implement raid array section 11.8 convert application read write request coordinated set disk o operation implement sophisticated error handling data recovery algorithm take step optimize o functionality implement device hard- ware device driver application software observe progression depict figure 12.17 initially implement experimental o algorithm application level application code flexible application bug unlikely cause system crash furthermore develop code applica- tion level avoid need reboot reload device driver change code application level implementation inefficient overhead context switch application advantage internal kernel data structure kernel functionality efficient kernel messaging threading locking fuse system interface example allow file system write run user mode application level algorithm demonstrate worth reimplement kernel improve performance devel- opment effort challenging operate system kernel large complex software system kernel implementa- tion thoroughly debug avoid data corruption system high performance obtain specialized imple- mentation hardware device controller disad- vantage hardware implementation include difficulty expense make improvement fix bug increased devel- device controller code hardware device code hardware increase development cost increase time generation device functionality progression o performance storage network latency opment time month day decrease flexibility instance hardware raid controller provide mean kernel influence order location individual block read write kernel special information workload enable improve o performance time aspect computing o device increase speed nonvolatile

o performance time aspect computing o device increase speed nonvolatile memory device grow popularity variety device available speed nvm device vary high extraordinary generation device near speed dram development increase pressure o subsystem operate system algorithm advantage read write speed available figure 12.18 show cpu storage device dimension capacity latency o operation add figure representation networking latency reveal performance tax networking add o.

basic hardware element involve o bus device controller device work move datum device main memory perform cpu program o offload dma controller kernel module control device device driver system- interface provide application design handle basic category hardware include block device character stream device memory map file network socket program interval timer system call usually block process issue nonblock- e asynchronous call kernel application sleep wait o operation complete kernel o subsystem provide numerous service o scheduling buffering caching spooling device reservation error han- dling service translation make connection hardware device symbolic file name application involve level mapping translate character string name specific device driver device address phys- ical address o port bus controller mapping occur file system space unix separate device space ms dos streams implementation methodology provide frame- work modular incremental approach write device driver network protocol streams driver stack datum pass sequentially bidirectionally processing o system call costly term cpu consumption layer software physical device application layer imply overhead source context switching cross kernel protection boundary signal interrupt handling service o device load cpu memory system copy datum kernel buffer application space state advantage place functionality device controller kernel state disadvantage example handshaking section 12.2 bit busy bit command ready bit possible implement handshaking bit describe protocol explain bit insufficient system use interrupt drive o manage single serial port poll o manage end processor terminal poll o completion waste large number cpu cycle processor iterate busy wait loop time o complete o device ready service polling efficient catch dispatch interrupt describe hybrid strategy combine polling sleeping interrupt o device service strategy pure polling pure interrupt hybrid describe compute environment strategy efficient dma increase system concurrency complicate important scale system bus device speed cpu distinguish driver end stream module streams

hennessy patterson 2012 describe multiprocessor system cache- consistency issue intel 2011 good source information intel pro- detail pcie find https://pcisig.com acpi use fuse user mode file system create performance prob- lem analysis issue find https://www.usenix.org hennessy patterson 2012 j. hennessy d. patterson computer archi- tecture quantitative approach fifth edition morgan kaufmann 2012 intel 64 ia-32 architectures software developer manual com- bin volume 1 2a 2b 3a 3b.

intel corporation 2011 chapter 12 exercise multiple interrupt different device appear time priority scheme determine order interrupt service discuss issue need consider assign priority different interrupt advantage disadvantage support memory- map o device control register consider following o scenario single user pc mouse graphical user interface tape drive multitaske operate system device disk drive contain user file graphic card direct bus connection accessible scenario design operate system use buffering spooling caching combination use poll o interrupt drive o reason choice multiprogrammed system user program access memory virtual address operate system use raw physi- cal address access memory implication design initiation o operation user program exe- cution operate system kind performance overhead associate service interrupt describe circumstance block o describe circumstance nonblocking o implement nonblocking o process busy wait device ready typically completion device o single interrupt raise appropriately handle host processor certain setting code execute completion o break separate piece piece execute imme- diately o complete schedule second interrupt remain piece code execute later time purpose strategy design interrupt handler dma controller support direct virtual memory access target o operation specify virtual address translation virtual physical address perform dma design complicate design dma controller advantage provide functionality unix coordinate activity kernel o component manip- ulate share kernel datum structure windows use object- orient message passing kernel o component discuss pro con approach write pseudocode implementation virtual clock include queueing management timer request kernel application assume hardware provide timer channel discuss advantage disadvantage guarantee reliable transfer datum module stream abstraction file collection related information define creator file map operate system physical mass storage device file system describe file map physical device access manipulate user program access physical storage slow file system design efficient access requirement important include provide support file sharing remote access c h p t e r user file

remote access c h p t e r user file system visible aspect general purpose operating system provide mechanism line storage access datum program operate system user computer system file system consist distinct part collection file store related datum directory structure organize provide information file system file system live storage device describe chapter 11 continue discuss chapter chapter consider aspect file major directory structure discuss semantic share file multiple process user computer finally discuss way handle file protection necessary multiple user want control access file file access explain function file system describe interface file system discuss file system design tradeoff include access method file shar- ing file locking directory structure explore file system protection 13.1 file concept computer store information storage medium nvm device hdd magnetic tape optical disk computer system convenient use operate system provide uniform logical view store information operate system abstract physical property storage device define logical storage unit fil file map operate system physical device storage device usually nonvolatile content persistent system reboot file name collection related information record sec- ondary storage user perspective file small allotment logical secondary storage datum write secondary storage file commonly file represent program source object form datum data file numeric alphabetic alphanu- meric binary file free form text file format rigidly general file sequence bit byte line record mean- ing define file creator user concept file extremely general file method user application use store retrieve datum general purpose use stretch original confine example unix linux oper- ating system provide proc file system use file system interface provide access system information process detail information file define creator different type information store file source executable program numeric text datum photo music video file certain define struc- ture depend type atext fil sequence character organize line possibly page source fil sequence function organize

character organize line possibly page source fil sequence function organize declaration follow executable statement executable fil series code section loader bring memory execute file name convenience human user refer usually string character example.c system differentiate uppercase lowercase character name system file name independent process user system create instance user create file example.c user edit file specify file owner write file usb drive send e mail attachment copy network call example.c destination system sharing synchonization method second copy independent change separately file attribute vary operating system typically consist symbolic file information keep human- identifie unique tag usually number identify file file system non human readable file type information need system support different type location information pointer device location file device size current size file byte word block possibly maximum allow size include attribute protection access control information determine reading writing executing timestamp user identificatio information keep creation modification use datum useful pro- tection security usage monitoring new file system support extended file attribute include char- acter encoding file security feature file checksum figure 13.1 illustrate fil info window macos display file attribute information file keep directory structure reside device file typically directory entry consist file unique identifier identifier turn locate file attribute kilobyte record information file system file size directory megabyte gigabyte directory match volatility file like file store device usually bring memory piecemeal need file info window macos file abstract data type define file properly need consider operation perform file operate system provide system call create write read reposition delete truncate file let examine operate system perform seven basic file operation easy similar operation rename file implement create fil step necessary create file space file system find file discuss allocate space file chapter 14 second entry new file open fil file operation specify file cause operate system evaluate check access

file operation specify file cause operate system evaluate check access permis- sion operation create delete require file open successful open return file handle argument call write fil write file system specify open file handle information write file system write pointer location file write place sequential write pointer update write occur read fil read file use system specify file handle memory block file system need read pointer location file read place sequential read take place read pointer update process usually read write file current operation location keep process current file positio pointer read write operation use pointer save space reduce system reposition file current file position pointer open file reposition give value repositioning file need involve actual o. file operation know file seek delete fil delete file search directory name file having find associate directory entry release file space reuse file erase mark free directory entry note system allow hard link multiple name direc- tory entry file case actual file content delete link delete truncate fil user want erase content file attribute force user delete file recreate function allow attribute remain unchanged file length file reset length zero file space release seven basic operation comprise minimal set require file operation common operation include append new information end exist file rename existing file primitive operation combine perform file operation instance create copy file create new file read old write new want operation allow user set attribute file example want operation allow user determine status file file length set file attribute file owner mention file operation mention involve search directory entry associate name file avoid constant searching system require open system file operate system keep table call open fil table contain information open file file operation request file specify index table searching require file long actively close process operate system remove entry open

file long actively close process operate system remove entry open file table potentially release lock create delete system call work closed open file system implicitly open file reference file automatically close job program open file terminate system require programmer open file explicitly open system file open operation take file search directory copy directory entry open file table open accept access- mode information create read read write append mode check file permission request mode allow file open process open system typically return pointer entry open file table pointer actual file o operation avoid searching simplify system interface implementation open close operation com- plicated environment process open file simulta- neously occur system different application open file time typically operate system use level internal table process table system wide table process table track file process open store table information process use file instance current file pointer file find access right file accounting information include entry process table turn point system wide open file table system wide table contain process independent information location file disk access date file size file open process system wide table include entry file process execute open new entry simply add process open file table point appropriate entry system-

wide table typically open file table open count associate file indicate process file open close decrease open count open count reach zero file long use file entry remove open file table file locking java java api acquire lock require obtain filechannel file lock lock method filechannel acquire lock api lock method filelock lock(long begin long end boolean share begin end beginning end position region lock set share true share lock set share false acquire lock exclusively lock release invoke release filelock return lock operation program figure 13.2 illustrate file locking java program acquire lock file file.txt lock half file exclusive lock lock second half share lock summary piece information associate open file file pointer system include file offset read write system call system track read write location current file position pointer pointer unique process operate file keep separate disk file attribute file open count file close operate system reuse open file table entry run space table multiple process open file system wait file close remove open file table entry file open count track number open close reach zero close system remove entry location fil file operation require system read write datum file information need locate file locate mass storage file server network ram drive keep memory system read directory structure operation access right process open file access mode informa- tion store process table operate system allow deny subsequent o request operate system provide facility lock open file sec- tion file file lock allow process lock file prevent process gain access file lock useful file share process example system log file access modify number process system file lock provide functionality similar reader writer lock cover section 7.1.2 ashared lock akin reader lock process acquire lock concurrently exclusive lock behave like writer lock process time acquire lock important note public class lockingexample

lock process time acquire lock important note public class lockingexample public static final boolean exclusive = false public static final boolean shared = true public static void main(string arg throw ioexception filelock sharedlock = null filelock exclusivelock = null randomaccessfile raf = new randomaccessfile("file.txt","rw channel file filechannel ch = raf.getchannel lock half file exclusive exclusivelock = ch.lock(0 raf.length()/2 exclusive modify datum release lock lock second half file share sharedlock = ch.lock(raf.length()/2 + 1,raf.length(),shared read datum release lock catch java.io ioexception ioe exclusivelock = null sharedlock = null file locking example java operate system provide type lock system provide exclusive file locking furthermore operate system provide mandatory advi- sory file locking mechanism mandatory locking process acquire exclusive lock operate system prevent process access lock file example assume process acquire exclusive lock file system.log attempt open

system.log

process example text editor operate system prevent access exclusive lock release alternatively lock advisory operate system prevent text editor acquire access sys- tem.log text editor write manually acquire lock access file word locking scheme manda- tory operate system ensure locking integrity advisory locking software developer ensure lock appropriately acquire release general rule windows operate system adopt mandatory locking unix system employ advisory lock use file lock require precaution ordinary process syn- chronization example programmer develop system manda- tory locking careful hold exclusive file lock access file prevent process access file furthermore measure take ensure process involve deadlock try acquire design file system entire operate system consider operate system recognize support file type operate system recognize type file operate file reasonable way example common mistake occur user try output binary object form program attempt normally produce garbage attempt succeed operate system tell file binary object program common technique implement file type include type file split part extension usually separate period figure 13.3 way user operate system tell type file operate system allow user specify file sequence character follow period terminate extension additional character example include resume.docx server.c system use extension indicate type file type operation file file .com .exe .sh extension execute instance .com .exe file form binary executable file .sh file shell script contain ascii format command operate system application program use extension indicate file type interested example java compiler expect source file .java extension microsoft word word processor expect file end .doc .docx extension extension require user specify file extension save typing application look file give extension expect extension support operate system consider hint application operate consider macos operating system system file type .app application file creator attribute exe com bin language link binary file contain audio v information mpeg mov mp3 relate file group file com- press archive

mpeg mov mp3 relate file group file com- press archive rar zip tar ascii binary file format print print view gif pdf jpg library routine lib dll command command textual datum document xml html tex source code c cc java perl common file type contain program create attribute set operate system create use enforce support system instance file produce word processor word processor creator user open file double click mouse icon represent file word processor invoke automatically file load ready edit unix system use magic number store beginning binary file indicate type datum file example format image file likewise use text magic number start text file indicate type file shell language script write detail magic number computer jargon http://www.catb.org/esr/jargon/. file magic number system feature base solely information unix record create program unix allow file name- extension hint extension enforce depend operate system mean aid user determine type content file contain extension ignore give application application programmer file type indicate internal structure file source object file structure match expectation program read certain file conform require structure understand operate system example operate system require executable file specific structure determine memory load file location instruction operate system extend idea set system support file structure set special operation manipulate file point bring disadvantage have operate system support multiple file structure make operate system large cumbersome operate system define different file structure need contain code support file structure addition necessary define file file type support operate system new application require information structure way support operate system severe problem result example assume system support type file text file compose ascii character separate carriage return line feed executable binary file user want define encrypt file protect content read unauthorized people find file type appropriate encrypt file ascii text line apparently random bit appear binary file executable result

text line apparently random bit appear binary file executable result circumvent misuse operate system file type mechanism abandon encryption scheme operate system impose support minimal number file structure approach adopt unix windows unix consider file sequence 8 bit byte interpretation bit operate system scheme provide maximum flexibility little support application program include code interpret input file appropriate structure operate system support structure executable file system able load run program internal file structure internally locate offset file complicated operate system disk system typically define block size determine size sector disk o perform unit block physical record block size unlikely physical record size exactly match length desire logical record logical record vary length pack number logical record physical block common solution problem example unix operating system define file simply stream byte byte individually addressable offset beginning end file case logical record size 1 byte file system automatically pack unpack byte physical disk block 512 byte block necessary logical record size physical block size packing technique deter- logical record physical block packing user application program operate system case file consider sequence block basic o function operate term block conversion logical record physical block relatively simple software problem disk space allocate block portion block file generally waste block 512 byte example file 1,949 byte allocate block 2,048 byte 99 byte waste waste incur unit block instead byte internal fragmentation file system suffer internal fragmentation large block size great internal 13.2 access method file store information information access read computer memory information file access way system provide access method file mainframe operate system support access method choose right particular application major design problem simple access method sequential access information file process order record mode access far common example editor compiler usually access file read write bulk operation file read operation read next()—reads portion file automatically bulk operation file read operation read next()—reads portion file automatically advance file pointer track o location similarly write operation write next()—append end file advance end newly write material new end file file reset beginning system program able skip forward backward n record integer n n = 1 sequential access depict figure 13.4 base tape model file work sequential access device random access one method direct access relative access file fixed length logical record allow program read write record rapidly particular order direct access method base disk model file disk allow random access file block direct access file view number sequence block record read write read block 14 read block 53 write block 7 restriction order read write direct access file direct access file great use immediate access large amount information database type query concern particular subject arrive compute block contain answer read block directly provide desire information simple example airline reservation system store information particular flight example flight 713 block identify flight number number available seat flight 713 store block 713 reservation file store information large set people compute hash function people name search small memory index determine block read direct access method file operation modify include block number parameter read(n n block number read write(n write alternative approach retain read write add operation position file(n n block number effect read(n position file(n read block number provide user operate system normally relative block number relative block number index relative beginning file relative block file 0 1 absolute disk address 14703 block 3192 second use relative block number allow operate system decide file place call allocation problem discuss chapter 14 help prevent user access portion file system file system start relative block number 0 start 1 system satisfy request record n file assum- e logical record length l request record n

file assum- e logical record length l request record n turn o request l byte start location l (n file assume record n = 0

logical record fix size easy read write delete record operate system support sequential direct access file system allow sequential file access allow direct access system

require file define sequential direct create file access manner consistent declaration easily simulate sequential access direct access file simply keep variable cp define current position show figure 13.5 simulate direct access file sequential access file extremely inefficient clumsy access method access method build direct access method method generally involve construction index file index like index book contain pointer block find record file search index use pointer access file directly find desire record implementation direct access simulation sequential access direct access file example retail price file list universal product code upcs item associate price record consist 10 digit upc 6 digit price 16 byte record disk 1,024 byte block store 64 record block file 120,000 record occupy 2,000 block 2 million byte keep file sort upc define index consist upc block index 2,000 entry 10 digit 20,000 byte keep memory find price particular item binary search index search learn exactly block contain desire record large file index file large keep memory solution create index index file primary index file contain pointer secondary index file point actual data master index point disk block secondary index secondary index block point actual file block file keep sort define key find particular item binary search master index provide block number secondary index block read binary search find block contain desire record finally block search sequentially way record locate key direct access read figure 13.6 show similar situation implement openvms index relative file 13.3 directory structure directory view symbol table translate file name file control block view directory organize way organization allow insert entry delete entry search name entry list entry directory section examine scheme define logical structure directory system consider particular directory structure need mind operation perform directory search fil need able search directory structure find entry particular file file symbolic name similar smith john

find entry particular file file symbolic name similar smith john social security age example index relative file name indicate relationship file want able find file name match particular pattern create fil new file need create add directory delete file file long need want able remove directory note delete leave hole directory structure file system method defragement directory list directory need able list file directory content directory entry file list rename file file represent content user able change content use file change rename file allow position directory structure change traverse file system wish access directory file directory structure reliability good idea save content structure entire file system regular interval copy file magnetic tape secondary storage network system cloud technique provide backup copy case system failure addition file long use file copy backup target disk space file release reuse file follow section describe common scheme define logical structure directory simple directory structure single level directory file con- taine directory easy support understand figure single level directory significant limitation number file increase system user file directory unique name user datum file test.txt unique rule violate example programming class 23 student call program second assignment prog2.c 11 call assign2.c fortunately file system support file name 255 character relatively easy select unique file name single user single level directory find difficult remem- ber name file number file increase uncommon user hundred file computer system equal number additional file system keep track file daunting task see single level directory lead confusion file name different user standard solution create separate directory user level directory structure user user fil direc- tory ufd ufds similar structure list file single user user job start user log system master fil directory mfd search mfd index user account number entry point ufd user figure 13.8 user refer particular file ufd search different

user figure 13.8 user refer particular file ufd search different user file long file name ufd unique create file user operate system search user ufd ascertain file user 1 user 2 user 3 user 4 level directory structure exist delete file operate system confine search local ufd accidentally delete user file user directory create delete necessary special system program run appropriate user account information program create new ufd add entry mfd execution program restrict system administra- tor allocation disk space user directory handle technique discuss chapter

14 file level directory structure solve collision prob- lem disadvantage structure effectively isolate user isolation advantage user completely independent disadvantage user want cooperate task access file system simply allow local user file access user access permit user ability file user directory particular file uniquely level directory user file level directory think tree inverted tree height 2 root tree mfd direct descendant ufds descendant ufds file file leave tree specify user file define path tree root mfd leaf specify file user file define path file system path file uniquely user know path file desire example user wish access test file name test.txt simply refer test.txt

access file name test.txt user b directory entry userb refer /userb test.txt system syntax name file direc- tory user additional syntax need specify volume file instance windows volume specify letter follow colon file specification c:userbtest system separate volume directory file
part speci- fication openvms instance file login.com specify u:[sst.crissmeyer]login.com;1 u volume sst directory crissmeyer subdirectory 1 version number system unix linux simply treat volume directory give volume rest directory file instance /u pgalvin test specify volume u directory pgalvin file test special instance situation occur system file program provide system loader assembler compiler utility rou- tine library generally define file appropriate command give operate system file read loader execute command interpreter simply treat command file load execute directory system define file search current ufd solution copy system file ufd copy system file waste enormous space system file require 5 mb support 12 user require 5 × 12 = 60 mb copy system file standard solution complicate search procedure slightly special user directory define contain system file example user 0 file give load operate system search local ufd file find find system automatically search special user directory contain system file sequence directory search file name call search path search path extend contain unlimited list directory search command give method unix windows system design user search path see view level directory level tree natural generalization extend directory structure tree arbitrary height figure 13.9 generalization allow user create subdirectory organize file accordingly tree

common directory structure tree root directory file system unique path directory subdirectory contain set file subdirectory implementation directory simply file treat special way directory internal format bit directory entry define entry file 0 subdirectory 1 special tree structured directory structure system call create delete directory case operate system file system code implement file format normal use process current directory current

code implement file format normal use process current directory current directory contain file current interest process reference file current directory search file need current directory user usually specify path change current directory directory hold file change directory system provide take directory parameter use redefine current directory user change current directory want system leave application shell track operate current directory process different current directory initial current directory user login shell designate user job start user log operate system search accounting file predefine location find entry user accounting purpose accounting file pointer user initial directory pointer copy local variable user specify user initial current directory shell process spawn current directory subprocess usually current directory parent spawn path name type absolute relative unix linux absolute path begin root designate initial follow path specify file give directory name path arelative path define path current directory example tree structure file system figure 13.9 current directory /spell mail relative path prt refer file absolute path /spell mail prt allow user define subdirectory permit impose structure file structure result separate directory file associate different topic example subdirectory create hold text book different form information example directory program contain source program directory bin store binary note executable file know system binary lead store bin directory interesting policy decision tree structured directory concern handle deletion directory directory entry directory contain simply delete suppose direc- tory delete contain file subdirectory approach take system delete directory delete directory user delete file directory subdirectory exist procedure apply recursively delete approach result substantial work alternative approach take unix rm command provide option request delete directory

directory file subdirectory delete approach fairly easy implement choice policy policy convenient dangerous entire directory structure remove command command issue error large number file directory need restore assume backup exist tree structure directory system user allow access addition

backup exist tree structure directory system user allow access addition file file user example user b access file user specify path user b specify absolute relative path alternatively user b change current directory user directory access file file consider programmer work joint project file asso- ciate project store subdirectory separate project file programmer programmer equally responsible project want subdirectory directory situation common subdirectory share share directory file exist file system place tree structure prohibit sharing file directory acyclic graph graph cycle allow directory share subdirec- tory file figure 13.10 file subdirectory different directory acyclic graph natural generalization tree- structure directory scheme important note share file directory copy file copy programmer view copy original programmer change file change appear copy share file actual file exist change person immediately visible sharing acyclic graph directory structure particularly important subdirectory new file create person automatically appear share subdirectory people work team file want share directory home directory team member contain directory share file subdirectory case single user user file organization require file place different subdirectory example program write particular project directory program directory project shared file subdirectory implement way common way exemplify unix system create new directory entry call link link effectively pointer file subdirectory example link implement absolute relative path reference file search directory directory entry mark link real file include link information resolve link path locate real file link easily identify format directory entry have special type system support type effectively indirect pointer operate system ignore link traverse directory tree preserve acyclic structure system common approach implement share file simply duplicate information share directory entry identical equal consider difference approach creation link link clearly different original directory entry equal duplicate directory entry original copy indistinguishable major problem duplicate directory entry maintain consistency file modify acyclic graph directory structure flexible simple tree structure

file modify acyclic graph directory structure flexible simple tree structure complex problem consider carefully file multiple absolute path name consequently distinct file name refer file situation similar aliasing problem programming language try traverse entire file system find file accumulate statistic file copy file backup storage problem significant want traverse share structure problem involve deletion space allocate share file deallocate reuse possibility remove file delete action leave dangle pointer nonexistent file bad remain file pointer contain actual disk address space subsequently reuse file dangle pointer point middle file system sharing implement symbolic link situation somewhat easy handle deletion link need affect original file link remove file entry delete space file deallocate leave link dangle search link remove list associate link keep file search expensive alternatively leave link attempt use time determine file give link exist fail resolve link access treat illegal file case system designer consider carefully file delete file create symbolic link original file case unix symbolic link leave file delete user realize original file go replace microsoft windows use approach approach deletion preserve file reference delete implement approach mechanism determine reference file delete list reference file directory entry symbolic link link copy directory entry establish new entry add file reference list link directory entry delete remove entry list file delete file reference list trouble approach variable potentially large size file reference list need entire list need count number reference add new link directory entry increment reference count delete link entry decrement count count 0 file delete remain reference unix operating system use approach nonsymbolic link hard link keep reference count file infor- mation block inode section c.7.2 effectively prohibit multiple reference directory maintain acyclic graph structure avoid problem one discuss system simply allow share directory link general graph directory problem acyclic graph structure ensure cycle start level directory allow problem acyclic graph structure ensure cycle start level directory allow user create subdirectory tree structured directory result fairly easy simply add new file subdirectory exist tree structured directory preserve tree structured nature add link tree structure destroy result simple graph structure figure primary advantage

acyclic graph relative simplicity algorithm traverse graph determine reference file want avoid traverse share section acyclic graph twice mainly performance reason search major share subdirectory particular file find want avoid search subdirectory second search waste time cycle allow exist directory likewise want avoid search component twice reason correctness perfor- mance apoorly design algorithm result infinite loop continually search cycle terminate solution limit arbitrarily number directory access search similar problem exist try determine file delete acyclic graph directory structure value 0 reference count mean reference file directory file delete cycle exist reference count 0 long possible refer directory file anomaly result possibility self reference cycle directory structure case generally need use garbage collection general graph directory scheme determine reference delete disk space reallocate garbage collection involve traverse entire file system mark access second pass collect mark list free space similar marking procedure ensure traversal search cover file system garbage collection disk base file system extremely time consume seldom attempt garbage collection necessary possible cycle graph acyclic graph structure easy work difficulty avoid cycle new link add structure know new link complete cycle algorithm detect cycle graph computationally expensive especially graph disk storage simple algorithm special case directory link bypass link directory traversal cycle avoid extra overhead incur information store computer system want safe physical damage issue reliability improper access issue reliability generally provide duplicate copy file comput- er system program automatically computer operator intervention copy disk file tape regular interval day week month maintain copy file system accidentally destroy file system damage hardware problem error read write power surge failure head crash dirt temperature extreme vandalism file delete accidentally bug file system soft- ware cause file content lose reliability cover detail chapter 11 protection provide way laptop system run modern operate system provide protection require user password authentication access encrypt secondary

system provide protection require user password authentication access encrypt secondary stor- age open laptop remove drive difficult time access datum firewalle network access use difficult break network connection multius system valid access system need advanced mechanism allow valid access datum type access need protect file direct result ability access file system permit access file user need protection provide complete protection prohibit access alternatively provide free access protection approach extreme general use need control access protection mechanism provide control access limit type file access access permit deny depend factor type access request different type operation control read read file write write rewrite file execute load file memory execute append write new information end file delete delete file free space possible reuse list list attribute file attribute change change attribute file operation rename copy edit file control system high level function implement system program make low level system call protection provide low level instance copy file implement simply sequence read request case user read access cause file copy print protection mechanism propose advantage disadvantage appropriate intended application small computer system member research group example need type protection large corporate computer research finance personnel operation discuss approach protection follow section present complete treatment chapter 17 common approach protection problem access depen- dent identity user different user need different type access file directory general scheme implement identity- dependent access associate file directory access control list acl specify user name type access allow user user request access particular file operate system check access list associate file user list request access access allow protection violation occur user job deny access file approach advantage enable complex access methodolo- gy main problem access list length want allow read file list user read access technique undesirable consequence construct list tedious unrewarding task especially know advance list user system directory

tedious unrewarding task especially know advance list user system directory entry previously fix size variable size result complicated space management problem resolve use condensed version access condense length access control list system recognize classification user connection file owner user create file owner group set user share file need similar access group work group user system common recent approach combine access control list general easy implement owner group universe access- control scheme describe example solaris use category access default allow access control list add specific file directory fine grained access control desire illustrate consider person sara write new book hire graduate student jim dawn jill help project text book keep file name book.tex protection associate file follow sara able invoke operation file jim dawn jill able read write file allow delete file user able read write file sara interested let people possible read text obtain feedback permission unix system unix system directory protection file protection handle similarly associate file directory field owner group universe consist bit rwx r control read access w control write access x control execution user list content subdirectory r bit set appropriate field similarly user change current directory current directory foo x bit associate foo subdirectory set sample directory listing unix environment show sep 3 08:30 jul 8 09.33 jul 8 09:35 aug 3 14:13 feb 24 2017 feb 24 2017 jul 31 10:31 aug 29 06:52 jul 8 09:35 field describe protection file directory d character indicate subdirectory show number link file owner group size file byte date modification finally file optional extension achieve protection create new group text member jim dawn jill group text associate file book.tex access right set accordance policy outline consider visitor sara like grant temporary access chapter 1 visitor add text group access chapter file group sara add group chapter 1

group access chapter file group sara add group chapter 1 addition access control- list functionality visitor add access control list scheme work properly permission access list con- troll tightly control accomplish way example unix system group create modify manager facility superuser control achieve human interaction access list discuss section 17.6.2 limited protection classification field need define protection field collection bit bit allow prevent access associate example unix system define field bit rwx r control read access w control write access x control execution separate field keep file owner file group user scheme bit file need record protection information example protection field file book.tex follow owner sara bit set group text r w bit set universe r bit set difficulty combine approach come user interface user able tell optional acl permission set file solaris example + append regular permission 19 -rw r r--+ 1

jim staff 130 25 22:13 file1 separate set command setfacl getfacl manage windows user typically manage access control list gui figure 13.12 show file permission window windows 7 ntfs file system example user guest specifically deny access file listpanel.java difficulty assign precedence permission acl conflict example walter file group read permission file acl grant walter read write permission write walter grant deny solaris operate system acl precedence fine grained assign default follow general rule specificity priority protection approaches approach protection problem associate password file access computer system control pass- word access file control way password choose randomly change scheme effective lim- ite access file use password disadvantage number password user need remember large make scheme impractical second password file discover file accessible protection basis system allow user associate password subdirectory individual file address problem commonly encryption partition individual file provide strong protection password management key multilevel directory structure need protect individual file collection file subdirectory need provide mechanism directory protection directory operation protect somewhat different file operation want control creation deletion file directory addition probably want control user determine existence file directory knowledge existence file significant list content directory protect operation similarly path refer file directory user allow access directory file system file numerous path name acyclic general graph give user different access right particular file depend path windows 10 access control list management 13.5 memory map file method access file commonly consider sequential read file disk standard system call open read write file access require system disk access alternatively use virtual memory technique discuss chapter 10 treat file o routine memory access approach

know memory map file allow virtual address space logically associate file shall lead significant memory mapping file accomplish map disk block page page memory initial access file proceed ordinary demand paging result page fault page sized portion file read file system physical page system opt read page sized chunk memory time subsequent read write file handle routine memory access manipulate file memory incur overhead read write system call simplify speed file access usage note write file map memory necessarily imme- diate synchronous write file secondary storage generally system update file base change memory image file close memory pressure system intermediate change swap space lose free memory use file close memory map datum write file secondary storage remove virtual memory process operate system provide memory mapping specific system use standard system call perform file o. system choose memory map file regardless file specify memory map let solaris example file specify memory map mmap system solaris map file address space process file open access ordinary system call open read write solaris memory map file file map kernel address space regardless file open solaris treat file o memory- map allow file access place efficient memory subsystem avoid system overhead cause traditional read multiple process allow map file concurrently allow sharing datum write process modify datum virtual memory see map section file give early discussion virtual memory clear sharing memory map section memory implement virtual memory map share process point page physical memory page hold copy disk block memory sharing illustrate figure 13.13 memory map system call support copy write functionality allow process share file read mode copy datum modify access share datum coordinate process involve use mechanism achieve mutual exclusion describe chapter 6 share memory fact implement memory map file scenario process communicate share mem- ory have communicate process memory map file virtual

share mem- ory have communicate process memory map file virtual address space memory map file serve region share memory communicate process figure 13.14 see section 3.5 posix share memory object create communicate process memory map object address space follow section discuss support windows api share memory memory map file share memory windows api general outline create region share memory memory- map file windows api involve create fil mapping file map establish view map file process virtual address space second process open create view map file virtual address space map file represent share memory object enable communication place illustrate step detail example producer process create share memory object memory map fea- ture available windows api producer write message share memory consumer process open mapping shared- memory object read message write consumer share memory memory map o.

establish memory map file process open file map createfile function return handle open file process create mapping file handle createfilemapping function file mapping process establish view map file virtual address space mapviewoffile function view map file represent por- tion file map virtual address space process entire file portion map sequence program int main(int argc char argv handle hfile hmapfile hfile = createfile("temp.txt file generic read | generic write read write access 0 sharing file null default security open open new existing file file attribute normal routine file attribute null file template hmapfile = createfilemapping(hfile file handle null default security page readwrite read write access map page 0 map entire file text("sharedobject name share memory object lpmapaddress = mapviewoffile(hmapfile map object handle file map access read write access 0 map view entire file write share memory sprintf(lpmapaddress,"shared memory message producer writing share memory windows api show figure 13.15 eliminate error checking code createfilemapping create name share memory object call sharedobject consumer process communicate share memory segment create mapping name object producer create view memory map file virtual address space pass parameter value 0 indicate map view entire file instead pass value specify offset size create view contain subsection file important note entire mapping load memory mapping establish map file demand page bring page memory access mapviewoffile function return pointer share memory object access memory location access memory map file instance producer process write message share memory message share memory program illustrate consumer process establish view name share memory object show figure 13.16 program int main(int argc char argv hmapfile = openfilemapping(file map access r w access false inheritance text("sharedobject map file object lpmapaddress = mapviewoffile(hmapfile map object handle file map access read write access 0

map view entire file read share memory printf("read message s lpmapaddress consumer reading share memory windows api somewhat simple show figure 13.15 necessary process create mapping exist name share memory object consumer process create view map file

producer process program figure 13.15 consumer read share memory message share memory message write producer process finally process remove view map file unmapviewoffile provide programming exercise end chapter share memory memory mapping windows api file abstract data type define implement operate system sequence logical record logical record byte line fixed variable length complex data item operate system specifically support record type leave support application program major task operate system map logical file concept physical storage device hard disk nvm device physical record size device logical record size necessary order logical record physical record task support operate system leave file system useful create directory allow file organize single level directory multiuser system cause naming problem file unique level directory solve problem create separate directory user file directory list file include file location disk length type owner time creation time use natural generalization level directory tree structure directory tree structured directory allow user create subdirectory organize file acyclic graph directory structure enable user share subdirectory file complicate searching deletion general graph structure allow complete flexibility sharing file direc- tory require garbage collection recover unused disk remote file system present challenge reliability performance security distributed information system maintain user host access information client server share state information man- age use access file main information storage mechanism computer system file protection need multiuser system access file control separately type access read write execute append delete list directory file protection provide access list password technique system automatically delete user file user log job terminate user explicitly request keep system file user explicitly delete discuss relative merit approach system track type file leave user simply implement multiple file type system well similarly system support type structure file datum simply support stream byte advantage disadvantage approach simulate multilevel directory structure single level directory structure arbitrarily long name

multilevel directory structure single level directory structure arbitrarily long name answer yes explain contrast scheme multilevel directory scheme answer explain prevent simulation success answer change file name limit seven character explain purpose open close operation system subdirectory read write autho- rize user ordinary file describe protection problem arise suggest scheme deal protection prob- consider system support 5,000 user suppose want allow 4,990 user able access file specify protection scheme unix suggest protection scheme effectively purpose scheme provide unix researcher suggest instead have access control list associate file specify user access file user control list associate user specify file user access discuss relative merit scheme

multilevel directory structure implement multics system organick 1972 operate system implement multilevel direc- tory structure include linux love 2010 macos singh 2007 solaris mcdougall mauro 2007 version windows russi- novich et al 2017 general discussion solaris file system find sun sys- tem administration guide devices file systems http://docs.sun.com/app/ network file system nfs design sun microsystems allow directory structure spread networked computer system nfs version 4 describe rfc3505 http://www.ietf.org/rfc/rfc3530.txt great source meaning computer jargon http://www.catb.org/ r. love linux kernel development edition developer mcdougall mauro 2007 r. mcdougall j. mauro solaris internals second edition prentice hall 2007 e. i. organick multics system examination struc- ture mit press 1972 russinovich et al 2017 m. russinovich d. a. solomon a. ionescu win- dows internals 1 seventh edition microsoft press 2017 a. singh mac os x internals systems approach

addison- chapter 13 exercise consider file system file delete disk space reclaim link file exist problem occur new file create storage area absolute path problem avoid open file table maintain information file currently open operate system maintain separate table user maintain table contain reference file currently access user file access different program user separate entry open file table explain advantage disadvantage provide mandatory lock instead advisory lock use leave user discretion provide example application typically access file accord following method system automatically open file reference time close file job terminate discuss advantage disadvantage scheme compare traditional user open close file explicitly operate system know certain application go access file datum sequential manner exploit information improve performance example application benefit operating- system support random access index file system provide file sharing maintain single copy file system maintain copy user share file discuss relative merit approach c h p t e r see chapter 13 file system provide mechanism on- line storage access file content include datum program file system usually reside permanently secondary storage design hold large datum chapter primarily concern issue surround file storage access common secondary storage medium hard disk drive nonvolatile memory device explore way structure file use allocate storage space recover free space track location datum interface part operate system secondary storage performance issue consider chapter give general purpose operating system provide file system additionally operating system allow administrator user add file system file system vary respect include feature performance reliability design goal different file system serve different purpose example temporary file system fast storage retrieval nonpersistent file default secondary storage file system linux ext4 sacrifice performance reliability feature see study operating system plenty choice variation make thorough coverage challenge chapter concentrate common denominator describe detail implement local file system directory struc- discuss block allocation free block

local file system directory struc- discuss block allocation free block algorithm trade off explore file system efficiency performance issue look recovery file system failure describe wafl file system concrete example 14.1 file system structure disk provide secondary storage file system main- taine characteristic convenient purpose 1 disk rewrite place possible read block disk modify block write block 2 disk access directly block information contain simple access file sequentially randomly switch file require drive move read write head wait medium rotate nonvolatile memory nvm device increasingly file storage location file system differ hard disk rewrite place different performance characteris- tic discuss disk nvm device structure detail chapter 11 improve o efficiency o transfer memory mass storage perform unit block block hard disk drive sector depend disk drive sector size usually 512 byte 4,096 byte nvm device usually block 4,096 byte transfer method similar disk drive file system provide efficient convenient access storage device allow datum store locate retrieve easily file system pose different design problem problem define file system look user task involve define file attribute operation allow file directory structure organize file second problem create algorithm datum structure map logical file system physical secondary storage device file system generally compose different level structure show figure 14.1 example layered design level design use feature low level create new feature use o control level consist device driver interrupt handler transfer information main memory disk system device driver think translator input consist high- level command retrieve block 123 output consist low level hardware specific instruction hardware controller interface o device rest system device driver usually write specific bit pattern special location o controller memory tell controller device location act action detail device driver o infrastructure cover chapter basic file system call block o subsystem linux need issue generic command appropriate device driver read

subsystem linux need issue generic command appropriate device driver read write block storage device issue command drive base logical block address concern o request scheduling layer manage memory buffer cache hold file- system directory datum block block buffer allocate transfer mass storage block occur buffer buffer manager find buffer memory free buffer space allow basic file system logical file system layer file system request o complete cache hold frequently file system metadata improve performance manage content critical optimum system performance module know file logical block file logical block number 0 1 n. file- organization module include free space manager track unal- locate block provide block file organization module finally logical file system manage metadata information metadata include file system structure actual datum content file logical file system manage directory structure provide file organization module information need give symbolic file maintain file structure file control block file control block fcb inode unix file system contain information file include ownership permission location file content logical file system responsible protection discuss chapter 13 17 layered structure file system implementation duplica- tion code minimize o control basic file system code multiple file system file system logical file system file organization module unfortunately layering introduce operating system overhead result decrease performance use layering include decision lay- er use layer major challenge design new file system use today operate system support example cd rom write iso 9660 for- mat standard format agree cd rom manufacturer addition removable medium file system operate system disk- base file system unix use unix fil system ufs base berkeley fast file system ffs windows support disk file system format fat fat32 ntfs windows nt file system cd rom dvd file system format linux support 130 different file system standard linux file system know extended file system common version ext3 ext4 distribute

system know extended file system common version ext3 ext4 distribute file system file system server mount client computer network file system research continue active area operating system design implementation google create file system meet company specific storage retrieval need include high- performance access client large number disk interesting project fuse file system provide flexibility file system development use implement execute file system user level kernel level code fuse user add new file system variety operate system use file system manage file 14.2 file system operation describe section 13.1.2 operate system implement open close system call process request access file content section delve structure operation implement file- storage memory structure implement file system structure vary depend operate system file system general principle apply storage file system contain information boot operate system store total number block number location free block directory structure individual file structure detail remainder chapter describe briefly boot control block volume contain information need system boot operate system volume disk contain operate system block typically block volume ufs call boot block ntfs partition boot sector volume control block volume contain volume detail number block volume size block free block count free block pointer free fcb count fcb pointer ufs call superblock ntfs store master fil table directory structure file system organize file ufs include file name associate inode number ntfs store master file table aper file fcb contain detail file unique identifier number allow association directory entry ntfs informa- tion actually store master file table use relational database structure row file memory information file system management performance improvement caching datum load mount time update file system operation discard dismount type structure include memory mount table contain information mount memory directory structure cache hold directory information recently access directory directory volume mount contain pointer volume table system wide open fil volume mount contain pointer volume table system wide open fil table contain copy fcb open file information process open fil table contain pointer appropriate entry system wide open file table information file process open buffer hold file system block read write file system create new file process call logical file system logical file system know format directory structure create new file allocate new fcb alternatively file system implementation create fcbs file system creation time fcb allocate set free fcb system read appropriate directory memory update new file fcb write file system atypical fcb show figure 14.2 operate system include unix treat directory exactly file type field indicate directory oper- file date create access write file owner group acl file datum block pointer file datum block typical file control block ate system include windows implement separate system call file directory treat directory entity separate file large structural issue logical file system file organization module map directory o storage block location pass basic file system o control system file create o.

open open pass file logical file system open system search system wide open file table file use process process open file table entry create point exist system wide open file table algorithm save substantial overhead file open directory structure search give file part directory structure usually cache memory speed directory operation file find fcb copy system wide open file table memory table store fcb track number process file open entry process open file table pointer entry system wide open file table field field include pointer current location file read write operation access mode file open open return pointer appropriate entry process file system table file operation perform pointer file open file table system use appropriate fcb locate disk cache save time subsequent open file give entry vary unix system refer fil descriptor windows refer process close file process table entry remove system wide entry open count decremente user open file close update metadata copy disk base directory structure system wide open file table entry remove caching aspect file system structure overlook system information open file actual data block memory bsd unix system typical use cache disk o save average cache hit rate

85 percent show technique worth implement bsd unix system describe fully appendix c. operating structure file system implementation summarize figure 14.3 14.3 directory implementation selection directory allocation directory management algorithm significantly affect efficiency performance reliability file sys- tem section discuss trade off involve choose open file memory file system structure file open b file read simple method implement directory use linear list file name pointer datum block method simple program time consume execute create new file search directory sure exist file add new entry end directory delete file search directory name file release space allocate reuse directory entry thing mark entry unused assign special blank assign invalid inode

thing mark entry unused assign special blank assign invalid inode number 0 include unused bit entry attach list free directory entry alternative copy entry directory free location decrease length directory link list decrease time require delete file real disadvantage linear list directory entry find file require linear search directory information frequently user notice access slow fact operating system implement software cache store recently directory information cache hit avoid need constantly reread information secondary storage sort list allow binary search decrease average search time requirement list keep sorted complicate create delete file substantial amount directory information maintain sort directory sophisticated tree datum structure balanced tree help advantage sorted list sorted directory listing produce separate data structure file directory hash table linear list store directory entry hash datum structure hash table take value compute file return pointer file linear list greatly decrease directory search time insertion deletion fairly straightforward provision collision situation file name hash major difficulty hash table generally fix size dependence hash function size example assume linear probe hash table hold 64 entry hash function con- vert file name integer 0 63 instance remainder division 64 later try create 65th file enlarge direc- tory hash table 128 entry result need new hash function map file name range 0 127 reorganize exist directory entry reflect new hash function value alternatively use chain overflow hash table hash entry link list instead individual value resolve collision add new entry link list lookup somewhat slow search require step link list collide table entry method likely fast linear search entire directory 14.4 allocation method direct access nature secondary storage give flexibility imple- mentation file case file store device main problem allocate space file storage space utilize effectively file access quickly major meth- od allocate secondary storage space wide use contiguous link index method advantage disadvantage

storage space wide use contiguous link index method advantage disadvantage system support common system use method file file system type contiguous allocation require file occupy set contiguous block device device address define linear ordering device ordering assume job access device access block b + 1 block b normally require head movement head movement need sector cylinder sector cylinder head need track hdd number disk seek require access contiguously allocate file contiguous allocation disk space minimal assume block close logical address close physically seek time seek finally need contiguous allocation file define address block length block unit file file n block long start location b occupy block b b + 1 b + 2 b + n 1 directory entry file indicate address start block length area allocate file figure 14.4 contiguous allocation easy implement limitation modern file system access file allocate contiguously easy sequential access file system remember address block reference necessary read block direct access block file start block b immediately access block b + i.

sequential direct access support contiguous allocation contiguous allocation problem difficulty find- e space new file system choose manage free space determine task accomplish management system discuss section 14.5 management system slow contiguous allocation problem see particular application general dynamic storage allocation problem discuss section 9.2

involve satisfy request size n list free hole fit good fit common strategy select free hole set available hole simulation show fit good fit efficient bad fit term time storage utilization fit good fit clearly good term storage utilization fit generally fast algorithm suffer problem external fragmentation file allocate delete free storage space break little piece external fragmentation exist free space break chunk problem large contiguous chunk insufficient request storage fragment number hole large store datum depend total disk storage average file size external fragmentation minor major problem strategy prevent loss significant amount storage space external fragmentation copy entire file system device original device free completely create large contiguous free space copy file original device allocate contiguous space large hole scheme effectively compact free space contiguous space solve fragmentation problem cost compaction time cost particularly high large storage device compact device hour necessary weekly basis system require function line file system unmounte time normal system operation generally permit compaction avoid cost production machine modern system need defragmentation perform line normal system operation performance penalty substantial problem contiguous allocation determine space need file file create total space need find allocate creator program person know size file create case determina- tion fairly simple copy exist file example general size output file difficult estimate allocate little space file find file extend especially well fit allocation strategy space side file use file large place possibility exist user program terminate appropriate error message user allocate space run program repeat run costly prevent user normally overestimate space need result considerable waste space possibility find large hole copy content file new space release previous space series action repeat long space exist time consume user need inform explicitly happen system continue despite problem slowly total space need file know advance preallocation inefficient file grow slowly long period month

know advance preallocation inefficient file grow slowly long period month year allocate space final size space unused long time file large internal fragmentation minimize drawback operate system use modify contiguous allocation scheme contiguous chunk space allocate initially prove large chunk contiguous space know extent add location file block record location block count plus link block extent system owner file set extent size setting result inefficiency owner incorrect internal fragmentation problem extent large external fragmentation problem extent vary size allocate deallocate commercial symantec veritas file system use extent optimize performance veritas high performance replacement standard unix ufs link allocation solve problem contiguous allocation link allocation file link list storage block block scatter device directory contain pointer block file example file block start block 9 continue block 16 block 1 block 10 finally block 25 figure 14.5 block contain pointer block pointer available user block 512 byte size block address pointer require 4 byte user see block 508 byte create new file simply create new entry directory link allocation directory entry pointer block file pointer initialize null end list pointer value signify file size field set 0 write file cause free- space management system find free block new block write link end file read file simply read block follow pointer block block external fragmentation link allocation free block free space list satisfy request size file need declare file create file continue grow long free block available consequently necessary compact disk space link allocation disadvantage major problem effectively sequential access file find ith linked allocation disk space block file start beginning file follow pointer ith block access pointer require storage device read require hdd seek consequently inefficient support direct access capability link allocation file disadvantage space require pointer pointer require 4 byte 512 byte block 0.78 percent disk pointer information file require slightly

byte block 0.78 percent disk pointer information file require slightly space usual solution problem collect block multiple call cluster allocate cluster block instance file system define cluster block operate secondary storage device cluster unit pointer use small percentage file space method allow logical physical block mapping remain simple improve hdd throughput few disk head seek require decrease space need block allocation free list management cost approach increase internal fragmentation space waste cluster partially block partially random o performance suffer request small datum transfer large datum cluster improve disk access time algorithm file system problem link allocation reliability recall file link pointer scatter device consider happen pointer lose damage bug operate system software hardware failure result pick wrong pointer error turn result link free space list file partial solution use doubly link list store file relative block number block scheme require overhead file important variation link allocation use file allocatio table fat simple efficient method disk space allocation ms dos operating system section storage beginning volume set aside contain table table entry block index block number fat way link list directory entry contain block number block file table entry index block number contain block number block file chain continue reach block special end file value table entry unused block indicate table value 0

allocate new block file simple matter find 0 value table entry replace previous end- file value address new block 0 replace end file value illustrative example fat structure show figure 14.6 file consist disk block 217 618 339

fat allocation scheme result significant number disk head seek fat cache disk head start volume read fat find location block question location block bad case move occur block benefit random access time improve disk head find location block read information number disk block 1 linked allocation solve external fragmentation size declaration prob- lem contiguous allocation absence fat link alloca- tion support efficient direct access pointer block scatter block disk retrieve order index allocation solve problem bring pointer location index block file index block array storage block address ith entry index block point ith block file directory contain address index block figure 14.7 find read ith block use pointer ith index block entry scheme similar paging scheme describe section 9.3 file create pointer index block set null ith block write block obtain free space manager address ith index block entry indexed allocation support direct access suffer external fragmentation free block storage device satisfy request space indexed allocation suffer waste space pointer overhead index block generally great pointer overhead link allocation consider common case file block link allocation lose space pointer block index allocation entire index block allocate pointer non null point raise question large index block file index block want index block small index allocation disk space possible index block small able hold pointer large file mechanism available deal issue mechanism purpose include follow link scheme index block normally storage block read write directly allow large file link index block example index block contain small header give file set 100 disk block address address word index block null small file pointer index block large file multilevel index avariant link representation use level index block point set

second level index block turn point file block access block operate system use level index find second level index block use block find desire data block approach continue fourth level depend desire maximum file size 4,096 byte block store 1,024 byte pointer index block level index allow 1,048,576 datum block file size 4 gb combined scheme alternative unix base file system 15 pointer index block file inode 12 pointer point direct block contain address block contain datum file datum small file 12 block need separate index block block size 4 kb 48 kb datum access directly pointer point indirect block point single indirect block index block contain datum address block contain datum second point double indirect block contain address block contain address block contain pointer actual datum block pointer contain address triple indirect block unix inode show figure 14.8 unix inode method number block allocate file exceed space addressable 4 byte file pointer operate system 32 bit file pointer reach 232 byte 4 gb unix linux implementation support 64 bit file pointer allow file file system exbibyte size zfs file system support 128 bit file pointer index allocation scheme suffer performance problem link allocation specifically index block cache memory data block spread volume allocation method discuss vary storage efficiency data block access time important criterion select proper method method operate system implement select allocation method need determine system system sequential access use method system random access type access contiguous allocation require access block easily initial address file memory calculate immediately address ith block block read directly link allocation address block memory read directly method fine sequential access direct access access ith block require block read problem indicate link allocation application require direct access result system support direct access file contiguous allocation sequential access file link allocation system type access declare

allocation sequential access file link allocation system type access declare file create file create sequential access link direct access file create direct access contiguous support direct access sequential access maximum length declare create case operate system appropriate data structure algorithm support allocation method file convert type creation new file desire type content old file copy old file delete new file rename indexed allocation complex index block mem- ory access directly keep index block memory require considerable space memory space avail- able read index block desire datum block level index index block read necessary extremely large file access block near end file require read index block need datum block finally read performance index allocation depend index structure size file position block desire system combine contiguous allocation index allocation contiguous allocation small file block auto- matically switch index allocation file grow large file small contiguous allocation efficient small file average performance good optimization use give disparity cpu speed disk speed unreasonable add thousand

extra instruc- tion operate system save disk head movement fur- thermore disparity increase time point hundred thousand instruction reasonably optimize head move- nvm device disk head seek different algorithm optimization need old algorithm spend cpu cycle try avoid nonexistent head movement inefficient existing file system modify new one create attain maximum performance nvm storage device development aim reduce instruction count overall path storage device application access datum 14.5 free space management storage space limit need reuse space delete file new file possible write optical disk allow write give sector reuse physically possible track free disk space system maintain free space list free space list record free device block allocate file directory create file search free space list require space allocate space new file space remove free space list file delete space add free space list free space list despite necessarily implement list discuss frequently free space list implement bitmap bit vector block represent 1 bit block free bit 1 block allocate bit 0 example consider disk block 2 3 4 5 8 9 10 11 12 13 17 18 25 26 27 free rest block allocate free space bitmap main advantage approach relative simplicity effi- ciency find free block n consecutive free block disk computer supply bit manipulation instruction effectively purpose technique find free block system use bit vector allocate space sequentially check word bitmap value 0 0 value word contain 0 bit represent set allocate block non-0 word scan 1 bit location free block calculation block number number bit word × number 0 value word + offset 1 bit hardware feature drive software functionality unfortu- nately bit vector inefficient entire vector

keep main memory write device contain file system occasionally recov- ery need keep main memory possible small device necessarily large one 1.3 gb disk 512 byte block need bitmap 332 kb track free block cluster block group reduce number 83 kb disk 1 tb disk 4 kb block require 32 mb 240 212 = 228 bit = 225 byte = 25 mb store bitmap give disk size constantly increase problem bit vector continue escalate approach free space management link free block keep pointer free block special location file system cache memory block contain pointer free block recall early example section 14.5.1 block 2 3 4 5 8 9 10 11 12 13 17 18 25 26 27 free rest block allocate situation pointer block 2 free block block 2 contain pointer block 3 point block 4 point block 5 point block 8 figure 14.9 scheme efficient traverse list read block require substantial o time hdd fortunately traverse free list frequent action usually free space list head linked free space list disk operate system simply need free block allocate block file block free list fat method incorporate free block accounting allocation datum structure separate method modification free list approach store address n free block free block n1 block actually free block contain address n free block address large number free block find quickly unlike situation standard link list approach approach take advantage fact generally contigu- ous block allocate free simultaneously particularly space allocate contiguous allocation algorithm clustering keep list n free block address address free block number n free contiguous block follow block entry free space list consist device address count entry require space simple disk address overall list short long count generally great 1 note method track free space similar extent method allocate block entry store balanced tree link list

extent method allocate block entry store balanced tree link list efficient lookup insertion deletion oracle zfs file system find

solaris operate system design encompass huge number file directory file system zfs create file system hierarchy scale meta- datum o large performance impact consider example free space list implement bitmap bitmap modify block allocate free free 1 gb datum 1 tb disk cause thousand block bitmap update data block scatter entire disk clearly data structure system large inefficient management free space zfs use combination technique control size datum structure minimize o need manage structure zfs create metaslab divide space device chunk manageable size give volume contain hundred metaslab metaslab associate space map zfs use counting algorithm store information free block write count structure disk use log structured file system technique record space map log block activity allocate free time order count format zfs decide allocate free space metaslab load associated space map memory balance tree structure efficient operation index offset replay log structure memory space map accurate representa- tion

allocate free space metaslab zfs condense map possible combine contiguous free block single entry finally free space list update disk transaction orient operation zfs collection sorting phase block request occur zfs satisfy request log essence log plus balanced tree free list trim unused blocks hdds storage medium allow block overwrite update need free list manage free space block need treat specially free free block typically keep datum file pointer block datum overwrite block storage device allow overwrite nvm flash base storage device suffer badly algorithm apply recall section 11.1.2 device erase write erase large chunk block compose page relatively long time compare read new mechanism need allow file system inform storage device page free consider erasure block con- taine page entirely free mechanism vary base storage controller ata attach drive trim nvme base storage unallocate command specific controller command mechanism keep storage space available write capabil- ity storage device get need garbage collection block erasure lead decrease storage o write performance know write cliff trim mechanism similar capability garbage collection erase step occur device nearly allow device provide consistent performance 14.6 efficiency performance discuss block allocation directory- management option consider effect performance efficient storage use disk tend represent major bottleneck system performance slow main computer component nvm device slow compare cpu main memory performance optimize section discuss variety technique improve efficiency performance secondary storage efficient use storage device space depend heavily allocation directory algorithm use instance unix inode preallocate volume disk percentage space lose inode preallocate inode spread volume improve file system performance improve performance result unix allocation free space algorithm try file datum block near file inode block reduce seek time example let reconsider clustering scheme discuss section 14.4 improve file seek file transfer performance cost internal fragmentation reduce fragmentation bsd unix vary cluster size file grow large cluster

fragmentation bsd unix vary cluster size file grow large cluster fill small cluster small file cluster file system describe appendix c. type datum normally keep file directory inode entry require consideration commonly write date record supply information user determine file need back system access date user determine file read result keep information file read field directory structure write mean block read memory section change block write device operation secondary storage occur block cluster chunk time file open reading fcb read write requirement inefficient frequently access file weigh benefit performance cost design file system generally data item associate file need consider effect efficiency performance consider instance efficiency affect size pointer access datum system use 32 bit 64 bit pointer through- operate system 32 bit pointer limit size file 232 4 gb 64 bit pointer allow large file size 64 bit pointer require space store result allocation free space management method link list index use storage space efficienc performance difficulty choose pointer size fix allo- cation size operate system plan effect change fat file system support 32 mb fat entry 12 bit point 8 kb cluster disk capacity increase large disk split 32 mb partition file system track block 32 mb hard disk capacity 100 mb common disk data structure algorithm ms dos modify allow large file system fat entry expand 16 bit later 32 bit initial file system decision efficiency reason how- advent ms dos version 4 million computer user inconvenience switch new large file system solaris zfs file system use 128 bit pointer theoretically need extend minimum mass device capable store 2128 byte atomic level storage 272 trillion kilogram example consider evolution solaris operate sys- tem originally data structure fix length allocate system startup structure include process table open file table process table process create file table file open system

table process table process create file table file open system fail provide service user table size increase recompile kernel reboot system later release solaris modern linux kernel kernel structure allocate dynamically eliminate artificial limit system performance course algo- rithm manipulate table complicated operate system little slow dynamically allocate deallocate table entry price usual general functionality basic file system algorithm select improve performance way discuss chapter 12 storage device controller include local memory form board cache large store entire track block time hdd seek perform track read disk cache start sector disk head reduce latency time disk controller transfer sector request operate system block disk controller main memory operate system cache block system maintain separate section main memory buffer cache block keep assumption shortly system cache file datum page cache page cache use virtual memory technique cache file datum page file system orient block cache file datum virtual address far efficient cache physical disk block access interface virtual memory file system system include solaris linux windows use page caching cache process page file datum know unifie virtual memory version unix linux provide unifie buffer cache illustrate benefit unified buffer cache consider alternative read write o unified buffer cache open access file approach use memory mapping section 13.5 second use standard system call read write unified buffer cache situation similar figure 14.10 read write system call buffer cache memory mapping require cache page cache buffer cache memory mapping proceed read disk block file system store buffer cache virtual memory system interface buffer cache content file buffer cache copy page cache situation know double caching require cache file system datum twice waste memory waste significant cpu o cycle extra data movement system memory addition inconsistency cache result corrupt file contrast unified buffer cache provide memory mapping read write system call use page cache

provide memory mapping read write system call use page cache benefit avoid double caching allow virtual memory system manage file system datum unified buffer cache show figure 14.11 regardless cache storage block page recently lru section 10.4.4 reasonable general purpose algorithm block page replacement evolution solaris page caching algorithm reveal difficulty choose algorithm solaris allow process page cache share unused memory version early solaris 2.5.1 distinction allocate page process allocate page cache result system perform o operation available memory cache page high rate o page scanner section 10.10.3 reclaim page process page cache free memory run low solaris 2.6 solaris 7 optionally implement priority paging page scanner give priority process page page cache solaris 8 apply fix limit process page file system page cache prevent- efficienc performance read write o unified buffer cache e force memory solaris 9 10 change algorithm maximize memory use minimize thrashing issue affect performance o write file system occur synchronously asynchronously synchronous write occur order storage subsystem receive write buffer calling routine wait datum reach drive proceed asynchronous write datum store cache control return caller write asynchronous metadata write synchronous operate system frequently include flag open system allow process request write perform synchronously example database use feature atomic transaction assure datum reach stable storage require order system optimize page cache different replacement algorithm depend access type file afile read write sequentially page replace lru order recently page instead sequential access optimize technique know free read ahead free remove page buffer soon page request previous page likely waste buffer space read ahead request page subsequent page read cache page likely request current page process retrieve datum disk transfer cache save considerable time think track cache controller eliminate need read ahead multiprogrammed system high latency overhead involve make small transfer track cache main memory perform read ahead

make small transfer track cache main memory perform read ahead remain beneficial page cache file system device driver interest- e interaction small amount datum write file page buffer cache storage device driver sort output queue accord device address action allow disk driver minimize disk head seek synchronous write require process write disk simply write cache system asynchronously write datum disk convenient user process see fast write datum read disk file block o system read ahead write nearly asynchronous read output disk file system fast input small transfer counter intuition matter buffering caching available large continuous o overrun capacity end bottle- neck device performance consider write large movie file hdd file large page cache page cache available process page cache fill o occur drive speed current hdd read fast write instance performance aspect reverse small o performance file directory keep main memory storage volume care take ensure system failure result loss datum datum inconsistency system crash cause inconsistency storage file system datum structure directory structure free block pointer free fcb pointer file system apply change structure place typical operation create file involve structural change file system disk directory structure modify fcbs allocate data block allocate free count block decrease change interrupt crash inconsistency structure result example free fcb count indicate fcb allocate directory structure point fcb compound problem caching operate system optimize o performance change directly storage cache cache change reach storage device crash occur corruption possible addition crash bug file system implementation device con- troller user application corrupt file system file system vary method deal corruption depend file system datum structure algorithm deal issue cause corruption file system detect problem correct detection scan metadata file system confirm deny consistency system unfortunately scan minute hour occur time system boot alternatively file system record state file system metadata start metadata change status

system record state file system metadata start metadata change status bit set indicate metadata flux update metadata complete successfully file system clear bit status bit remain set consistency checker run consistency checker system program fsck unix compare datum directory structure metadata state storage try fix inconsistency find allocation free space management algorithm dictate type problem checker find successful fix instance link allocation link block block entire file reconstruct datum block directory structure recreate contrast loss directory entry index allocation system disastrous data block knowledge reason unix file system cache directory entry read write result space allocation metadata change synchronously correspond data block write course problem occur synchronous write interrupt crash nvm storage device contain battery supercapacitor provide power power loss write datum device buffer storage medium datum lose precaution protect corruption crash log structured file systems computer scientist find algorithm technology originally area equally useful area case database log base recovery algorithm logging algorithm apply successfully problem consistency checking result implementation know log base transaction orient journaling note consistency checking approach discuss pre- ceding section essentially allow structure break repair recovery problem approach inconsistency irreparable consistency check able recover structure result loss file entire directory consistency checking require human intervention resolve conflict inconvenient human available system remain unavail- able human tell proceed consistency checking take system clock time check terabyte datum hour clock time solution problem apply log base recovery technique file system metadata update ntfs veritas file system use method include recent version ufs solaris fact common file system include ext3 ext4 zfs fundamentally metadata change write sequentially log set operation perform specific task transaction change write log consider commit system return user process allow continue execution log entry replay actual file- system structure change pointer update indicate action complete incomplete entire commit transaction complete entry log indicate log

complete incomplete entire commit transaction complete entry log indicate log file actually circular buffer circular buffer write end space continue beginning overwrite old value go want buffer write datum save scenario avoid log separate section file system separate storage device system crash log file contain zero transaction transaction contain complete file system commit operate system complete transaction execute pointer work complete file system structure remain consistent problem occur transaction abort commit system crash change transaction apply file system undo preserve consistency file system recovery need crash eliminate problem benefit logging disk metadata update update proceed fast apply directly on- disk datum structure reason find performance advantage sequential o random o. costly synchronous random metadata write turn costly synchronous sequential write log structured file system logging area change turn replay asynchronously random write appropriate structure overall result significant gain performance metadata orient operation file creation deletion hdd storage alternative consistency checking employ network appli- ance wafl file system solaris zfs file system system overwrite block new datum transaction write datum meta- datum change new block transaction complete metadata structure point old version block update point new block file system remove old pointer old block available reuse old pointer block keep snapshot create snapshot view file system specific point time update time apply solution require consistency checking pointer update atomically wafl consistency checker failure scenario cause metadata corruption section 14.8 detail wafl file system zfs take innovative approach disk consistency like wafl overwrite block zfs go provide checksum- ming metadata datum block solution combine raid assure datum correct zfs consistency checker detail zfs find section 11.8.6 backup restore storage device fail care take ensure datum lose failure lose forever end system program datum storage device magnetic tape secondary storage device recovery loss individual file entire device matter restore datum minimize copying need

individual file entire device matter restore datum minimize copying need use information file directory entry instance backup program know example wafl file system backup file file write date directory indicate file change date file need copy typical backup schedule follow day 1 copy backup medium file file system call backup day 2 copy medium file change day 1 day 3 copy medium file change day 2 day n. copy medium file change day n1 day 1 new cycle backup write previous set new set backup medium method restore entire file system start restore backup continue incremental backup course large value n great number medium read complete restore add advantage backup cycle restore file accidentally delete cycle retrieve delete file backup previous day length cycle compromise backup need number day cover restore decrease number tape read restore option perform backup day file change backup way restore recent incremental backup backup incremental backup need trade file modify day successive incremental backup involve file backup medium user notice particular file miss corrupt long damage reason usually plan backup time time save forever good idea store permanent backup far away regular backup protect hazard fire destroy computer backup tv mr. robot hacker attack primary source bank datum backup site have multiple backup site bad idea datum important 14.8 example wafl file system secondary storage o huge impact system performance file system design implementation command lot attention system designer file system general purpose provide reasonable performance functionality wide variety file size file type o load optimize specific task attempt provide well performance area general purpose file system write file layout

wafl netapp inc. example sort optimization wafl powerful elegant file system optimize random write wafl exclusively network file server produce netapp mean use distribute file system provide file client nfs cifs iscsi ftp http protocol design nfs cifs client use protocol talk file server server large demand random read large demand random write nfs cifs protocol cache datum read operation write great concern file server creator wafl file server include nvram cache write wafl designer take advantage run specific architecture optimize file system random o stable storage cache ease use guide principle wafl creator design include new snapshot functionality create multiple read copy file system different point time shall file system similar berkeley fast file system modification block base use inode describe file inode contain 16 pointer block indirect block belong file describe inode file system root inode metadata live file inode file free block map free inode map show figure 14.12 standard file data block limit location place file system expand addition disk length metadata file automatically expand file system

wafl file system tree block root inode base snapshot wafl create copy root inode file metadata update new block overwrite exist block new root inode point metadata datum change result write snapshot old root inode point old block update provide access file system instant snapshot take little storage space essence extra space occupy snapshot consist block modify snapshot free block map free inode map file file system wafl file layout example wafl file system important change standard file system free block map bit block bitmap bit set snapshot block snapshot block delete bitmap block zero block free reuse block overwrite write fast write occur free block nearest current head location performance optimization wafl snapshot exist simultaneously take hour day day month example user access snapshot access file time snapshot take snapshot facility useful backup testing versioning wafl snapshot facility efficient require copy write copy datum block take block modify file system provide snapshot frequently efficiency wafl snapshot depict figure 14.13 new version wafl actually allow read write snapshot know clone clone efficient technique shapshot case read snapshot capture state file system clone refer read snapshot write clone store new block clone pointer update refer new block original snapshot unmodified give view file system snapshot b snapshot block change c block d change d ´ snapshot wafl apple file system 2017 apple inc. release new file system replace 30 year old hfs+ file system hfs+ stretch add new feature usual process add complexity line code add feature difficult start scratch blank page allow design start current technology methodology provide exact set feature need apple file system apfs good example design goal run current apple device apple watch iphone mac computer create file system work watchos o tvos macos certainly challenge apfs feature rich include 64 bit pointer clone file directory snapshot

feature rich include 64 bit pointer clone file directory snapshot space sharing fast directory sizing atomic safe save primitive copy write design encryp- tion single- multi key o coalescing understand nvm hdd storage feature discuss new concept worth explore space sharing zfs like feature storage avail- able large free space container file system draw allocation allow apfs format volume grow shrink fast directory sizing provide quick space calculation updating atomic safe save primitive available api file system com- mand perform rename file bundle file directory single atomic operation o coalescing optimization nvm device small write gather large write optimize write performance apple choose implement raid new apfs instead depend exist apple raid volume mechanism software raid apfs compatible hfs+ allow easy conversion exist clone update clone promote replace original file system involve throw old pointer associate old block clone useful testing upgrade original version leave untouched clone delete test upgrade fail feature naturally result wafl file system implemen- tation replication duplication synchronization set datum network system snapshot wafl file system duplicate system snapshot take source system relatively easy update remote system send block contain new snapshot block one change time snapshot take remote system add block file system update pointer new system duplicate source system time second snapshot repeat process maintain remote system nearly date copy system replication disaster recovery system destroy datum available use remote system finally note zfs file system support similarly efficient snapshot clone replication feature common file system time go file system reside secondary storage design hold large datum permanently common secondary storage medium disk use nvm device increase storage device segment partition control medium use allow multiple possibly vary file system single device file system mount logical file system architecture available use file system implement layered modular structure low level deal physical property storage device

layered modular structure low level deal physical property storage device communicate upper level deal symbolic file name logical property file file file system allocate space storage device way contiguous link index allocation contiguous allocation suffer external fragmentation direct access inefficient link allocation index allocation require substantial overhead index block algorithm optimize way contiguous space enlarge extent increase flexibility decrease external fragmentation index allocation cluster multiple block increase throughput reduce number index entry need indexing large cluster similar contiguous allocation extent free space allocation method influence efficiency disk space use performance file system reliability secondary storage method include bit vector link list optimiza- tion include grouping counting fat place link list contiguous area directory management routine consider efficiency performance reliability hash table commonly method fast efficient unfortunately damage table system crash result inconsistency directory information disk content consistency checker repair damage file system struc- ture operate system backup tool allow datum copy magnetic tape storage device enable user recover datum loss entire device loss hardware failure operating system bug user error fundamental role file system play system operation performance reliability crucial technique log struc- ture caching help improve performance log structure raid improve reliability wafl file system example optimiza- tion performance match specific o load consider file currently consist 100 block assume file control block index block case index alloca- tion memory calculate disk o operation require contiguous link index single level alloca- tion strategy block following condition hold contiguous allocation case assume room grow beginning room grow end assume block information add store memory block add beginning block add middle block add end block remove beginning block remove middle block remove end bit map file allocation keep mass storage main memory consider system support strategy contiguous linked index allocation criterion decide strategy well utilize particular file problem contiguous allocation user

decide strategy well utilize particular file problem contiguous allocation user preallo- cate space file file grow large space allocate special action take solution problem define file structure consist initial con- tiguous area specify size area fill operate sys- tem automatically define overflow area link initial contiguous area overflow area fill overflow area allocate compare implementation file standard contiguous link implementation cache help improve performance system use large cache useful advantageous user operate system dynami- cally allocate internal table penalty operating internal bsd unix system cover mckusick et al 2015 detail concern file system linux find love 2010 google file system describe ghemawat et al 2003 fuse find http://fuse.sourceforge.net log structured file organization enhance performance con- sistency discuss rosenblum ousterhout 1991 seltzer et al 1993 seltzer et al 1995 log structure design networked file system propose hartman ousterhout 1995 thekkath et al zfs source code space map find http://src.opensolaris.o rg source xref onnv onnv gate usr src uts common/ fs zfs space map.c zfs documentation find

http://www.opensolaris.org/os/commu ntfs file system explain solomon 1998 ext3 file system linux describe mauerer 2008 wafl file system cover hitz et al 1995 ghemawat et al 2003 s. ghemawat h. gobioff s.-t. leung google file system proceedings acm symposium operating systems hartman ousterhout 1995 j. h. hartman j. k. ousterhout zebra striped network file system acm transactions computer systems volume 13 number 3 1995 page 274–310 hitz et al 1995 d. hitz j. lau m. malcolm file system design nfs file server appliance technical report netapp 1995 r. love linux kernel development edition developer w. mauerer professional linux kernel architecture john wiley sons 2008 mckusick et al 2015 m. k. mckusick g. v. neville neil r. n. m. wat- son design implementation freebsd unix operating system second edition pearson 2015 rosenblum ousterhout 1991 m. rosenblum j. k. ousterhout design implementation log structured file system proceedings acm symposium operating systems principles 1991 page 1–15 seltzer et al 1993 m. i. seltzer k. bostic m. k. mckusick c. staelin implementation log structured file system unix usenix winter 1993 page 307–326 seltzer et al 1995 m. i. seltzer k. a. smith h. balakrishnan j. chang s. mcmains v. n. padmanabhan file system logging versus clustering performance comparison usenix winter 1995 page 249–264 d. a. solomon inside windows nt second edition microsoft thekkath et al 1997 c. a. thekkath t. mann

e. k. lee frangipani scalable distributed file system symposium operating systems principles 1997 page 224–237 chapter 14 exercise consider file system use modify contiguous allocation scheme support extent file collection extent extent correspond contiguous set block key issue system degree variability size extent advantage disadvantage follow scheme extent size size predetermine extent size allocate dynamically extent fix size size predeter- contrast performance technique allocate disk block contiguous link index sequential ran- dom file access advantage variant link allocation use fat chain block file consider system free space keep free space list suppose pointer free space list lose system reconstruct free space list explain answer consider file system similar unix index allocation disk o operation require read content small local file /a b c assume disk block currently cache suggest scheme ensure pointer lose result memory failure file system allow disk storage allocate different level granularity instance file system allocate 4 kb disk space single 4 kb block 512 byte block advantage flexibility improve performance modification free space management scheme order support feature discuss performance optimization file system result difficulty maintain consistency system event computer crash discuss advantage disadvantage support link file cross mount point file link refer file store different volume consider file system disk logical physical block size 512 byte assume information file memory allocation strategy con- tiguous link index answer question logical physical address mapping accomplish system index allocation assume file 512 block long currently logical block 10 block access block 10 want access logical block 4 physical block read disk consider file system use inode represent file disk block 8 kb size pointer disk block require 4 byte file system 12 direct disk block single double triple indirect disk block maximum size file store file system fragmentation storage device eliminate com- paction typical disk device relocation base

storage device eliminate com- paction typical disk device relocation base register memory compact relocate file reason compact relocate file avoid explain log metadata update ensure recovery file sys- tem file system crash consider following backup scheme day 1 copy backup medium file disk day 2 copy medium file change day 1 day 3 copy medium file change day 1 differ schedule give section 14.7.4 have subsequent backup copy file modify backup benefit system section 14.7.4 drawback restore operation easy difficult explain answer discuss advantage disadvantage associate remote file system store file server set failure semantic different associate local file system implication support unix consistency semantic share access file store remote file system c h p t e r see chapter 13 file system provide mechanism line storage access file content include datum program chapter primarily concern internal structure operation file system explore detail way structure file use allocate storage space recover free space track location datum interface part operate system secondary storage delve detail file system implementation explore booting file sharing describe remote file system nfs example 15.1 file systems certainly general purpose computer store file typically thousand million billion file computer file store random access storage device include hard disk drive optical disk nonvolatile memory device see precede chapter general purpose computer system multiple storage device device slice partition hold volume turn hold file system depend- e volume manager volume span multiple partition figure 15.1 show typical file system organization computer system vary number file system file system vary type example typical solaris system dozen file system dozen different type show file system list figure 15.2 book consider general purpose file system worth note special purpose file system consider type file system solaris example mention typical storage device organization tmpf temporary file system create volatile main memory content erase

tmpf temporary file system create volatile main memory content erase system reboot crash objf virtual file system essentially interface kernel look like file system give debugger access kernel symbol ctfs virtual file system maintain contract information man- age process start system boot continue run lofs loop file system allow file system access place procf virtual file system present information process file system uf zf general purpose file system file system computer extensive file system useful segregate file group manage act group organization involve use directory section 14.3 15.2 file system mounting file open file system mount available process system specifically directory structure build multiple file system contain volume mount available file- system space mount procedure straightforward operate system give device mount point location file structure file system attach operate system require file system type provide inspect structure device solaris file system determine type file system typically mount point directory instance unix system file system contain user home directory mount /home access directory structure file system precede directory name /home /home jane mount file system /users result path /users jane use reach directory operate system verify device contain valid file system ask device driver read device directory verify directory expected format finally operate system note directory structure file system mount specify mount point scheme enable operate system traverse directory structure switch file system file system vary type appropriate illustrate file mounting consider file system depict figure 15.3 triangle represent subtree directory interest figure 15.3(a show exist file system figure 15.3(b show unmounted volume reside /device dsk point file exist file system access figure 15.4 show effect mount volume reside /device dsk /user volume unmounte file system restore situation depict figure 15.3 system impose semantic clarify functionality example system disallow mount directory contain file file system exist system b unmounte volume mount

contain file file system exist system b unmounte volume mount file system available directory obscure directory exist file file system unmounte terminate use file system allow access original file directory example system allow file system mount repeatedly different mount point allow mount file system consider action macos operate system system encounter disk time boot time system run macos operate system search file system device find automatically mount file system /volume directory add folder icon label file system store device directory user able click icon display newly mount file system microsoft windows family operate system maintain extended level directory structure device volume assign drive letter volume general graph directory structure associate volume mount /user partition mounting drive letter path specific file take form drive- letter:pathtofile recent version windows allow file system mount directory tree unix windows operate system automatically discover device mount locate file system boot time system like unix mount command explicit system configuration file contain list device mount point automatic mounting boot time mount execute manually issue concern file system mounting discuss section 15.3 section c.7.5 15.3 partition mounting layout disk variation depend operate system volume management software disk slice multiple partition volume span multiple partition multiple disk layout discuss appropriately consider form raid cover section 11.8 partition raw contain file system cooked contain file system raw disk file system appropriate unix swap space use raw partition example use format disk use file system likewise database use raw disk format datum suit need raw disk hold infor- mation need disk raid system bitmap indicate block mirror change need mirror similarly raw disk contain miniature database hold raid configuration information disk member raid set raw disk use discuss section 11.5.1 partition contain file system bootable properly instal configure operating system partition need boot information describe section 11.5.2 information format boot

partition need boot information describe section 11.5.2 information format boot time system file system code load interpret file system format boot information usually sequential series block load image memory execution image start predefine location byte image bootstrap loader turn know file system structure able find load kernel start execute boot loader contain instruction boot spe- cific operating system instance system dual boot allow- e install multiple operate system single system system know boot boot loader understand multi- ple file system multiple operate system occupy boot space load boot operate system available drive drive multiple partition contain different type file system different operating system note boot loader understand particular file system format operating system store file system bootable reason file system support root file system give operating system root partition select boot loader contain operating system kernel system file mount boot time volume automatically mount boot manually mount later depend operate system successful mount operation operate system verify device contain valid file system ask device driver read device directory verify directory expect format format invalid partition consistency check possibly correct user intervention finally operate system note memory mount table file system mount type file system detail function depend operate system microsoft windows base system mount volume separate space denote letter colon mention early record file system mount f example operate system place pointer file system field device structure correspond f process specify driver letter operate system find appropriate file system pointer traverse directory structure device find specify file directory later version windows mount file system point exist directory structure unix file system mount directory mounting imple- mente set flag memory copy inode directory flag indicate directory mount point field point entry mount table indicate device mount mount table entry contain pointer superblock file system device scheme enable operating system traverse directory superblock file system device scheme enable operating system traverse directory structure switch seamlessly file system vary type 15.4 file sharing ability share file desirable user want collaborate reduce effort require achieve computing goal user- orient operating system accommodate need share file spite inherent difficulty section examine aspect file sharing begin discuss general issue arise multiple user share file multiple user allow share file challenge extend sharing multiple file system include remote file system discuss challenge finally consider conflicting action occur share file instance multiple user write file write allow occur operate system protect user action operate system accommodate multiple user issue file sharing file naming file protection preeminent give directory structure allow file share user system mediate file sharing system allow user access file user virtual file systems default require user specifically grant access file issue access control protection cover section 13.4 implement sharing protection system maintain file directory attribute need single user system approach take meet requirement system evolve use concept file directory owner user group owner user change attribute grant access control file group attribute define subset user share access file example owner file unix system issue operation file member file group execute subset operation user execute subset operation exactly operation execute group member user definable file owner owner group id give file directory store file attribute user request operation file user id compare owner attribute determine request user owner file likewise group id compare result indicate permission applicable system apply permission request operation allow deny system multiple local file system include volume single disk multiple volume multiple attach disk case id checking permission matching straightforward file system mount consider external disk move system id system different care take sure id match system device file ownership reset occur example create new user id set file portable disk id

example create new user id set file portable disk id sure file accidentally accessible exist user 15.5 virtual file system see modern operating system concurrently support multiple type file system operate system allow multiple type file system integrate directory structure user seamlessly file system type navigate file system space discuss implementation detail obvious suboptimal method implement multiple type file system write directory file routine type instead operating system include unix use object orient technique sim- plify organize modularize implementation use method allow dissimilar file system type implement structure include network file system nfs user access file contain multiple file system local drive file system available network data structure procedure isolate basic system functionality implementation detail file system imple- mentation consist major layer depict schematically figure 15.5 layer file system interface base open read write close call file descriptor local file system local file system remote file system schematic view virtual file system second layer call virtual file system vfs layer vfs layer serve important function 1 separate file system generic operation implementation define clean vfs interface implementation vfs inter- face coexist machine allow transparent access different type file system mount locally 2 provide mechanism uniquely represent file net- work vfs base file representation structure call vnode contain numerical designator network wide unique file unix inode unique single file system network- wide uniqueness require support network file system kernel maintain vnode structure active node file direc- vfs distinguish local file remote one local file distinguish accord file system type vfs activate file system specific operation handle local request accord file system type call nfs protocol procedure protocol procedure network file system remote request file handle construct relevant vnode pass argu- ment procedure layer implement file system type remote file system protocol layer architecture let briefly examine vfs architecture linux main object

layer architecture let briefly examine vfs architecture linux main object type define linux vfs remote file systems inode object represent individual file fil object represent open file superblock object represent entire file system dentry object represent individual directory entry object type vfs define set operation implement object type contain pointer function table function table list address actual function implement define operation particular object example abbreviated api operation file object include int open .)—open file int close .)—close open file ssize t read .)—read file ssize t write .)—write file int mmap

.)—memory map file implementation file object specific file type require imple- ment function specify definition file object complete definition file object specify file struct file operation locate file /usr include linux fs.h vfs software layer perform operation object call appropriate function object function table have know advance exactly kind object deal vfs know care inode represent disk file directory file remote file appropriate function file read operation place function table vfs software layer function care datum actually 15.6 remote file systems advent network chapter 19 communication remote com- puter possible networking allow sharing resource spread campus world obvious resource share datum form file evolution network file technology remote file sharing method change implement method involve manually transfer file machine program like ftp second major method use distribute fil system dfs remote directory visible local machine way method world wide web reversion browser need gain access remote file separate operation essentially wrapper ftp transfer file increasingly cloud computing section 1.10.5 file sharing ftp anonymous authenticated access anonymous access allow user transfer file have account remote system world wide web use anonymous file exchange exclu- sively dfs involve tight integration machine access remote file machine provide file integration add complexity describe section client server model remote file system allow computer mount file system remote machine case machine contain file server machine seek access file client client server relationship common networked machine generally server declare resource available client specify exactly resource case file exactly client server serve multiple client client use multiple server depend implementation detail give client server facility server usually specify available file volume directory level client identification difficult client specify net- work identifier ip address spoof imitate result spoofing unauthorized client allow access server secure solution include secure authentication client encrypt key unfortunately security come challenge include ensure compatibility client encrypt key unfortunately security come challenge include ensure compatibility client server use encryption algorithm security key exchange intercept key allow unauthorized access difficulty solve problem unsecure authentication method commonly case unix network file system nfs authentication take place client networking information default scheme user id client server match server unable determine access right file consider example user id 1000 client 2000 server request client server specific file handle appropriately server determine user 1000 access file base determination real user id 2000 access grant deny base incorrect authentication information server trust client present correct user id note nfs protocol allow relationship server provide file client fact give machine server nfs client client remote file system mount file operation request send behalf user network server dfs protocol typically file open request send id request user server apply standard access check determine user credential access file mode request request allow deny allow file handle return client appli- cation application perform read write operation file client close file access complete operate system apply semantic similar local file system mount use different semantic remote file systems distributed information systems client server system easy manage distribute information sys- tem know distributed naming service provide unified access information need remote computing domain system dns provide host network address translation entire internet dns widespread file contain information send e mail ftp networked host obviously methodol- ogy scalable dns discuss section 19.3.1 distribute information system provide user password user id group id space distribute facility unix system employ wide variety distribute information method sun microsystems oracle corporation introduce yellow page rename network information service nis industry adopt use centralize storage user name host name printer information like unfortunately use unsecure authentication method include send user password unencrypted clear text identify host ip address sun nis+ secure replacement nis complicated identify host ip address sun nis+ secure replacement nis complicated widely adopt case microsoft common internet file system cifs network information conjunction user authentication user password create network login server use decide allow deny access request file system authentication valid user name match machine machine nfs microsoft use active directory distribute naming structure provide single space user establish distribute naming facility client server authenticate user microsoft version kerberos network authentication protocol industry move use lightweight directory access protocol ldap secure distribute naming mechanism fact active directory base ldap oracle solaris major operate system include ldap allow employ user authentication system wide retrieval information availability printer conceivably distribute ldap directory organization store user resource information organization computer result secure single sign user enter authentication information access computer organi- zation ease system administration effort combine location information currently scatter file system different distribute information service local file system fail variety reason include failure drive contain file system corruption directory structure disk management information collectively call metadata disk controller failure cable failure host adapter failure user system administrator failure cause file lose entire directory volume delete failure cause host crash error condition display human intervention require repair damage remote file system failure mode complexity network system require interaction remote machine problem interfere proper operation remote file system case network network interrupt host interruption result hardware failure poor hardware configuration networking implementation issue network build resiliency include multiple path host single failure interrupt flow df consider client midst remote file system file open remote host activity perform directory lookup open file read write datum file close file consider partitioning network crash server schedule shutdown server suddenly remote file system long reachable scenario common appropriate client system act local file system lose system terminate operation lost server delay operation server reachable failure semantic define implement lost server delay operation server reachable failure semantic define implement remote file system protocol termination operation result user lose datum patience dfs protocol enforce allow delaying file system operation remote host hope remote host available implement kind recovery failure kind state infor- mation maintain client server server client maintain knowledge current activity open file seamlessly recover failure situation server crash recognize remotely mount export file system open file nfs version 3 take simple approach implement stateless dfs essence assume client request file read write occur file system remotely mount file previously open nfs protocol carry information need locate appropriate file perform request operation similarly track client export volume mount assume request come legitimate stateless approach make nfs resilient easy implement make unsecure example forge read write request allow nfs server issue address industry standard nfs version 4 nfs stateful improve security performance 15.7 consistency semantic consistency semantic represent important criterion evaluate file system support file sharing semantic specify multiple user system access share file simultaneously particular specify modification datum user observable user semantic typically implement code file system consistency semantic directly relate process synchronization algorithm chapter 6 complex algorithm chapter tend implement case file o great latency slow transfer rate disk network example perform atomic transaction remote disk involve network communication disk read write system attempt set functionality tend perform poorly asuccessful implementation complex sharing semantic find andrew file system follow discussion assume series file access read write attempt user file enclose open close operation series access open close operation make fil session illustrate concept sketch prominent example consistency semantic unix file system chapter 19 use following consistency semantic write open file user visible immediately user file open mode sharing allow user share pointer current location file advancing pointer user affect share user file single image interleave access regardless origin

affect share user file single image interleave access regardless origin unix semantic file associate single physical image access exclusive resource contention single image cause delay user process andrew file system openafs use following consistency semantic write open file user visible immediately user file open file close change visible session start- e later open instance file reflect change accord semantic file associate temporarily possibly different image time consequently multiple user allow perform read write access concurrently image file delay constraint enforce scheduling unique approach immutable share file file declare share creator modify immutable file key property reuse content alter immutable file signify content file fix implementation semantic distribute system chapter 19 simple sharing discipline read network file system commonplace typically integrate overall directory structure interface client system nfs good example widely implement client server network file system use example explore implementation detail network file system nfs implementation specification software system access remote file lan wan nfs onc+ unix vendor pc operate system support implementa- tion describe solaris operating system modify version unix svr4 use tcp udp ip protocol depend interconnect network specification implementation intertwine description nfs detail need refer solaris implementation description general apply specification multiple version nfs late version 4 describe version 3 version commonly deploy nfs view set interconnected workstation set independent machine independent file system goal allow degree sharing file system explicit request transparent manner sharing base client server relationship machine client server sharing allow pair machine ensure machine independence sharing remote file system affect client machine machine remote directory accessible transparent manner particular machine m1 client machine carry mount operation semantic operation involve mount remote directory directory local file system mount oper- ation complete mount directory look like integral subtree local file system replace subtree descend local directory local directory root newly mount directory specification remote directory

directory local directory root newly mount directory specification remote directory argument mount operation transparently location host remote directory provide user machine m1 access file remote directory totally transparent manner illustrate file mounting consider file system depict figure 15.6 triangle represent subtree directory interest figure show independent file system machine name u s1 s2 point machine local file access figure 15.7(a show effect mount s1:/usr shared u:/usr local figure depict view user u file system mount complete access file dir1 directory prefix /usr local dir1 original directory /usr local machine long visible subject access right accreditation file system directory file system mount remotely local directory independent file system diskless workstation mount root server cascading mount permit nfs implementation file system mount file system remotely mount local machine affect mount invoke mount remote file system client access file system chance mount file system mount mechanism exhibit transitivity property figure 15.7(b illustrate cascade mount figure

show result mount s2:/usr dir2 u:/usr local dir1 remotely mount s1 user access file dir2 u prefix /usr local dir1 shared file system mount user home directory machine network user log workstation home environment property permit user mobility design goal nfs operate heterogeneous environ- ment different machine operating system network architecture mounting nfs mount b cascading mount nfs specification independent medium independence achieve use rpc primitive build external data representa- tion xdr protocol implementation independent interface system heterogeneous machine file system properly interface nfs file system different type mount locally nfs specification distinguish service provide mount mechanism actual remote file access service accordingly separate protocol specify service mount protocol pro- tocol remote file access nfs protocol protocol specify set rpc rpc building block implement transparent remote file access mount protocol mount protocol establish initial logical connection server client solaris machine server process outside kernel perform protocol function mount operation include remote directory mount server machine store mount request map correspond rpc forward mount server run specific server machine server maintain export list specify local file system export mount name machine permit mount solaris list /etc dfs dfstab edit superuser specification include access right read simplify maintenance export list mount table distributed naming scheme hold information available appropriate client recall directory export file system mount remotely accredited machine component unit directory server receive mount request conform export list return client file handle serve key access file mount file system file handle contain informa- tion server need distinguish individual file store unix term file handle consist file system identifier inode number identify exact mount directory export file system server maintain list client machine correspond- ing currently mount directory list mainly administrative purpose instance notify client server go addition deletion entry list server state affect mount protocol usually system static mounting preconfiguration establish

state affect mount protocol usually system static mounting preconfiguration establish boot time /etc vfstab solaris layout modify addition actual mount procedure mount protocol include procedure unmount return export list nfs protocol nfs protocol provide set rpc remote file operation proce- dure support following operation search file directory read set directory entry manipulate link directory access file attribute read write file procedure invoke file handle remotely mount directory establish omission open close operation intentional prominent feature nfs server stateless server maintain infor- mation client access parallel unix open file table file structure exist server consequently request provide set argument include unique file identifier absolute offset inside file appropriate operation result- e design robust special measure need take recover server crash file operation idempotent purpose operation perform multiple time effect perform achieve idempotence nfs request sequence number allow server determine request duplicate miss maintain list client mention violate statelessness server list essential correct operation client server need restore server crash consequently include inconsistent datum treat hint implication stateless server philosophy result synchrony rpc modified datum include indirection status block commit server disk result return client client cache write block flush server assume reach server disk server write nfs datum synchronously server crash recovery invisible client block server manage client intact result performance penalty large advantage caching lose performance increase storage nonvolatile cache usually battery back memory disk controller acknowledge disk write write store nonvolatile cache essence host see fast synchronous write block remain intact system crash write stable storage disk single nfs write procedure guarantee atomic intermix write call file nfs protocol provide concurrency control mechanism awrite system break rpc write nfs write read contain 8 kb datum udp packet limit 1,500 byte result user write remote file datum intermix claim lock management inherently stateful service outside nfs provide locking solaris user

management inherently stateful service outside nfs provide locking solaris user advise coordinate access share file mechanism outside scope nfs type schematic view nfs architecture nfs integrate operate system vfs illustration architecture let trace operation open remote file handle follow example figure 15.8 client initiate operation regular system operating system layer map vfs operation appropriate vnode vfs layer identify file remote invoke appropriate nfs procedure rpc nfs service layer remote server reinjecte vfs layer remote system find local invoke appropriate file- system operation path retrace return result advantage architecture client server identical machine client server actual service server perform kernel thread path translation nfs involve parsing path /usr local dir1 file.txt separate directory entry component 1 usr 2 local 3 dir1 path translation break path component name perform separate nfs lookup pair component directory vnode mount point cross component lookup cause separate rpc server expensive path traversal scheme need layout client logical space unique dictate mount client perform efficient hand server path receive target vnode mount point encounter point mount point particular client stateless server unaware lookup fast directory lookup cache client hold vnode remote directory name cache speed reference file initial path directory cache discard attribute return server match attribute cache recall implementation nfs allow mount remote file system mount remote file system cascade mount client cascade mount server involve path traversal client lookup directory server mount file system client see underlie directory instead mount directory exception open close file correspondence regular unix system call file operation nfs protocol rpc remote file operation translate directly correspond rpc conceptually nfs adhere remote service paradigm practice buffering caching technique employ sake performance direct correspondence exist remote operation rpc instead file block file attribute fetch rpc cache locally future remote operation use cache datum subject consistency constraint cache file attribute inode information cache

datum subject consistency constraint cache file attribute inode information cache file block cache file open kernel check remote server determine fetch revalidate cache attribute cache file block corresponding cache attribute date attribute cache update new attribute arrive server cache attribute default discard 60 second read ahead delay write technique server client client free delay write block server confirm datum write disk delay write retain file open concurrently conflict mode unix semantic section 15.7.1 preserve tune system performance make difficult characterize consistency semantic nfs new file create machine visible 30 second furthermore write file site visible site file open reading new open file observe change flush server nfs provide strict emulation unix semantic session semantic andrew section 15.7.2 spite drawback utility good performance mechanism widely multi vendor distribute system operation general purpose operating system provide file system type special purpose general volume contain file system mount computer file- depend operate system file system space seamless mount file system integrate directory structure distinct mount file system have designation file system bootable system able start contain operate system boot loader run simple program able find kernel file system load start execution system contain multiple bootable partition let administrator choose run boot time system multi user provide method file shar- ing file protection frequently file directory include metadata owner user group access permission mass storage partition raw block o file system file system reside volume compose partition multiple partition work volume manager simplify implementation multiple file system operate system use layered approach virtual file system interface make access possibly dissimilar file system seamless remote file system implement simply program ftp web server client world wide web functionality client server model mount request user id authenticate prevent unapproved access client server facility natively share information distribute information system dns allow sharing pro- vide unified user

distribute information system dns allow sharing pro- vide unified user space password management system identification example microsoft cifs use active directory employ version kerberos network authentication protocol pro- vide set naming authentication service computer network file sharing possible consistency semantic model cho- sen implement moderate multiple concurrent access file semantic model include unix session immutable share file nfs example remote file system provide client seam- access directory file entire file system featured remote file system include communication protocol remote opera- tion path translation explain vfs layer allow operate system support mul- tiple type file system easily file system type give system unix linux system implement procfs file system determine use procf interface explore process space aspect process view interface information gather system lack procf system integrate mount file system root file system naming structure use separate naming method mount file system give remote file access facility ftp remote file system like nfs create internal bsd unix system cover mckusick et al 2015 detail concern file system linux find love 2010 network file system

nfs discuss callaghan 2000 nfs ver- sion 4 standard describe http://www.ietf.org/rfc/rfc3530.txt ouster- hout 1991 discuss role distribute state networked file system nfs unix file system ufs describe mauro mcdougall b. callaghan nfs illustrated addison wesley 2000 r. love linux kernel development edition developer mauro mcdougall 2007 j. mauro r. mcdougall solaris internals core kernel architecture prentice hall 2007 mckusick et al 2015 m. k. mckusick g. v. neville

neil r. n. m. wat- son design implementation freebsd unix operating system second edition pearson 2015 j. ousterhout role distribute state cmu computer science 25th anniversary commemorative r. f. rashid ed addison- chapter 15 exercise assume particular augmentation remote file access pro- tocol client maintain cache cache translation file name corresponding file handle issue account implement cache give mount file system write operation underway system crash power loss file system remount file system log structured b file system log structured operating system mount root file system automatically operating system require file system root security ensure authentication system user protect integrity information store system datum code physical resource computer system security system prevent unauthorized access malicious destruction alteration datum accidental introduction inconsistency protection mechanism control access system limit type file access permit user addition protection ensure process gain proper authorization oper- ating system operate memory segment cpu protection provide mechanism control access program process user resource define computer system mechanism provide mean specify con- trol impose means enforce c h p t e r protection security vital computer system distinguish concept following way security measure con- fidence integrity system datum preserve protection set mechanism control access process user resource define computer system focus security chapter address protection chapter 17 security involve guard computer resource unauthorized access malicious destruction alteration accidental introduction inconsistency computer resource include information store system datum code cpu memory secondary storage tertiary storage networking compose computer facility chapter start examine way resource accidentally purposely misuse explore key security enabler cryptography finally look mechanism guard detect attack discuss security threat attack explain fundamental encryption authentication hashing examine use cryptography computing describe countermeasure security attack 16.1 security problem application ensure security computer system worth considerable effort large commercial system contain payroll financial datum invite target thief system contain datum pertainpayroll financial datum invite target thief system contain datum pertain- e corporate operation interest unscrupulous competitor furthermore loss datum accident fraud seriously impair ability corporation function raw compute resource attractive attacker bitcoin mining send spam source anonymously attack system chapter 17 discuss mechanism operate system pro- vide appropriate aid hardware allow user protect resource include program datum mechanism work long user conform intend use access resource system secure resource access intend circumstance unfortunately total security achieve nonetheless mechanism security breach rare occurrence norm security violation misuse system categorize inten- tional malicious accidental easy protect accidental misuse malicious misuse protection mechanism core accident avoidance follow list include form acci- dental malicious security violation note discussion security use term intruder hacker attacker attempt breach security addition threat potential security violation discovery vulnerability attack attempt break security breach confidentialit type violation involve unauthorized reading datum theft information typically breach confiden- tiality goal intruder capture secret datum system data stream credit card information identity information identity theft unreleased movie script result directly money intruder embarrassment hack institution breach integrity violation involve unauthorized modification datum attack example result passing liability innocent party modification source code important commercial open source application breach availability violation involve unauthorized destruction datum attacker wreak havoc status bragging right gain financially website defacement common example type security breach theft service violation involve unauthorized use resource example intruder intrusion program install daemon system act file server denial service violation involve prevent legitimate use system denial service dos attack accidental original internet worm turn dos attack bug fail delay rapid spread discuss dos attack section 16.3.2 attacker use standard method attempt breach secu- rity common masquerading participant commu- nication pretend host person mas- querading attacker breach authentication correctness identification gain access normally allow common attack replay capture exchange datum replay attack consist malicious common attack replay capture exchange datum replay attack consist malicious fraudulent repeat valid datum transmission replay comprise entire attack example repeat request transfer money frequently message security problem modificatio attacker change datum communication sender knowledge consider damage request authentication legitimate user information replace unau- thorized user kind attack man middle attack attacker sit data flow communication masquerade sender receiver vice versa network communication man middle attack precede session hijacking active communication session intercept broad class attack aim privilege escalation system assign privilege user user user administrator generally system include set privilege user account system frequently privilege assign nonuser system user internet access web page log anonymous user service file transfer sender email remote system consider privilege privilege send email receive user system privilege escalation give attacker privilege suppose example email contain script macro execute exceed email sender privilege masquerading message modification mention escalate privilege example common type attack difficult detect prevent attack category suggest absolute protection system malicious abuse possible cost perpetrator sufficiently high deter intruder case denial of- service attack preferable prevent attack sufficient detect countermeasure take stream filtering add resource attack deny service legitimate user protect system security measure level 1 physical site site contain computer system physically secure entry intruder machine room terminal computer access target machine secure example limit access building reside lock desk sit 2 network contemporary computer system server mobile device internet things iot device networked networking provide means system access external resource provide potential vector unauthorized access system computer datum modern system frequently travel pri- vate lease line share line like internet wireless connection dial line intercept datum harmful break computer interruption communication constitute remote denial service attack diminish user use trust 3 operate system operate system build set appli- cation service comprise huge code base harbor vulnerability insecure default setting misconfiguration security

huge code base harbor vulnerability insecure default setting misconfiguration security bug potential problem operate system keep date continuous patching hardened"—configure modify decrease attack surface avoid penetration attack surface set point attacker try break system 4 application party application pose risk especially possess significant privilege application inherently malicious benign application contain security bug vast number party application disparate code basis virtually impossible ensure application layered security model show figure 16.1 layer model security like chain link vulnerabil- ity layer lead system compromise respect old adage security strong weak link hold true factor overlook human authorization perform carefully ensure allow trusted user access system authorized user malicious encourage let use access willingly dupe social engineering use deception persuade people confidential information type social engineering attack phishing legitimate look e mail web page mislead user enter confidential information take click link browser page email inadvertently download malicious payload compromise system security user computer usually pc end target valuable resource compromised system attack system lan user far see factor level model plus human factor take account security maintain fur- thermore system provide protection discuss great detail chapter 17 allow implementation security feature abil- ity authorize user process control access log activity impossible operate system implement secu- rity measure run securely hardware protection feature need support overall protection scheme example system memory layered model security protection secure new hardware feature allow system secure shall discuss unfortunately little security straightforward intruder exploit security vulnerability security countermeasure create deploy cause intruder sophisticated attack exam- ple spyware provide conduit spam innocent system discuss practice section 16.2 turn deliver phishing attack target cat mouse game likely continue security tool need block escalate intruder technique activity remainder chapter address security network operating system level security application physical human lev- el important scope text security operate system operate system imple- mente

important scope text security operate system operate system imple- mente way range password authentication guard virus detect intrusion start exploration security threat 16.2 program threats processes kernel mean accomplish work computer write program create breach security cause normal process change behavior create breach common goal attacker fact nonprogram security event goal cause program threat example useful log system authorization lot useful leave door daemon remote access tool rat provide information allow easy access original exploit block section describe common method program cause security breach note considerable variation naming convention security hole use common descriptive term malware software design exploit disable damage computer system way perform activity explore major variation section system mechanism allow program write user execute user program execute domain provide access right execute user user misuse right program act clandestine malicious manner simply perform state function call trojan horse pro- gram execute domain escalate privilege example consider mobile app purport provide benign functionality flashlight app surreptitiously access user contact message smuggle remote server classic variation trojan horse trojan mule program emulate login program unsuspecting user start log terminal computer web page notice apparently mistype password try successful happen authentication key password steal login emulator leave run computer attacker reach bad url emulator store away password print login error message exit user provide genuine login prompt type attack defeat have operate system print usage message end interactive session require nontrappable key sequence login prompt control alt delete combination modern windows operate system user ensure url right valid variation trojan horse spyware spyware accompany program user choose install frequently come freeware shareware program include commercial software spyware download ad display user system create pop browser window certain site visit capture information user system return central site installation innocuous program windows system result loading spyware daemon spyware contact central site give

system result loading spyware daemon spyware contact central site give message list recipient address deliver spam message user windows machine process continue user discover spyware frequently spyware discover 2010 estimate 90 percent spam deliver method theft service consider crime country fairly recent unwelcome development class malware steal information ransomware encrypt information target computer render inaccessible owner informa- tion little value attacker lot value owner idea force owner pay money ransom decryption key need decrypt datum dealing criminal course payment ransom guarantee return access trojan malware especially thrive case vio- lation principle privilege commonly occur operate system allow default privilege normal user need user run default administrator true windows operate system windows 7 case operate system immune system permission protection kind can- kick malware persist survive reboot extend reach locally network violate principle privilege case poor operating system design decision making operate system software gen- eral allow fine grained control access security privilege need perform task available task execution control feature easy manage understand inconvenient inadequate misunderstood security measure bind circum- vent cause overall weakening security design form malware designer program system leave hole software capable type security breach trap door door show movie war games principle privilege principle privilege program privileged user system operate privilege necessary complete job purpose principle reduce number potential interaction privileged program minimum necessary operate correctly develop confidence unintentional unwanted improper use privilege occur ”

—jerome h. saltzer describe design principle instance code check specific user id password circumvent normal security procedure receive id password programmer trap door method embezzle bank include rounding error code have occasional half cent credit account account crediting add large money consider number transaction large bank execute trap door set operate specific set logic condi- tion case refer logic bomb door type especially difficult detect remain dormant long time possi- bly year detect usually damage example network administrator destructive reconfiguration company network execute program detect long employ company clever trap door include compiler compiler generate standard object code trap door regardless source code compile activity particularly nefarious search source code program reveal problem reverse engineering code compiler reveal trap door type attack perform patch compiler compile time library fact 2015 malware target apple xcode compiler suite dub xcodeghost affect software developer compromise version xcode download directly trap door pose difficult problem detect analyze source code component system give soft- ware system consist million line code analysis frequently frequently software development methodology help counter type security hole code review code review developer write code submit code base developer review code approve pro- vide comment define set reviewer approve code comment address code resubmitte review code admit code base compile debug finally release use good software developer use development ver- sion control system provide tool code review example git https://github.com/git/ note automatic code review define buffer size 0 int main(int argc char argv int j = 0 char buffer[buffer size int k = 0 argc < 2 return -1 printf("k d j d buffer sn j k buffer c program buffer overflow condition code scan tool design find flaw include security flaw gen- erally

good programmer good code reviewer involve develop code code review useful find report flaw find exploit software source code available make code review hard software malicious nonetheless pose threat security code injection attack executable code add modify benign software harbor vulnerability exploit allow attacker program code subvert exist code flow entirely reprogramme supply new code code injection attack nearly result poor insecure programming paradigm commonly low level language c c++ allow direct memory access pointer direct mem- ory access couple need carefully decide size memory buffer care exceed lead memory corruption memory buffer properly handle example consider simple code injection vector buffer over- flow program figure 16.2 illustrate overflow occur unbounded copy operation strcpy function copy regard buffer size question halt null 0 byte encounter byte occur buffer size reach program behave expect copy easily exceed buffer answer outcome overflow depend largely length overflow overflow content figure 16.3 vary greatly code generate compiler optimize possible outcome buffer overflow way affect outcome optimization involve adjustment memory layout commonly repositioning padding variable 1 overflow small little buffer size good chance entirely unnoticed allocation buffer size byte pad architecture specify boundary commonly 8 16 byte padding unused memory overflow technically bound 2 overflow exceed padding automatic variable stack overwrite overflow content outcome depend exact positioning variable semantic example employ logical condition subvert uncontrolled overflow lead program crash unexpected value variable lead uncorrectable 3 overflow greatly exceed padding current function stack frame overwrite frame function return address access function return flow program subvert redirect attacker region memory include memory control attacker example input buffer stack heap inject code execute allow attacker run arbitrary code process effective id note careful programmer perform bound checking size argv[1 strncpy function strcpy replace line strcpy(buffer argv[1 strncpy(buffer argv[1 sizeof(buffer)-1 unfortunately good bound checking exception norm strcpy know class

sizeof(buffer)-1 unfortunately good bound checking exception norm strcpy know class vulner- able function include sprintf gets function regard buffer size size aware variant harbor vulnerability couple arithmetic operation finite length integer lead integer overflow point danger inherent simple oversight maintain buffer clearly evident brian kerningham dennis ritchie book c programming language refer possible outcome undefined behavior perfectly predictable behavior coerce attacker demonstrate morris worm document rfc1135 https://tools.ietf.org/html/rfc1135 year later article issue 49 phrack magazine smash stack fun profit http://phrack.org/issues/49/14.html introduce exploitation technique masse unleash deluge exploit achieve code injection injectable code attacker write short code segment follow void func void execvp("/bin sh /bin sh null execvp system code segment create shell process program attack run root permission newly create shell gain complete access system course code segment allow privilege attack process code segment compile assembly binary opcode form transform binary stream compile form refer shellcode classic function spawn shell term grow encompass type code include advanced code add new user system reboot connect network wait remote instruction call reverse shell shellcode exploit show figure 16.4 code briefly redirect execution location like trampoline bounce code flow spot trampoline code execution exploit buffer overflow fact shellcode compiler metasploit project notable example care specific ensure code compact contain null byte case exploitation string copy terminate null compiler mask shellcode alphanumeric character attacker manage overwrite return address func- tion pointer vtable take simple case redirect address point supply shellcode commonly load user input environment variable file network input assume mitigation exist describe later shellcode execute hacker succeed attack alignment consideration handle add sequence nop instruction shellcode result know nop sle cause execution slide nop instruction payload encounter execute example buffer overflow attack reveal considerable knowl- edge programming skill need recognize exploitable code exploit unfortunately great programmer launch security attack hacker determine bug write exploit

great programmer launch security attack hacker determine bug write exploit rudimentary computer skill access exploit call script kiddie try launch attack target system buffer overflow attack especially pernicious run system travel allowed communication channel attack occur protocol expect communicate target machine hard detect prevent bypass security add firewall section 16.6.6 note buffer overflow vector manipulate code injection overflow exploit occur heap memory buffer free over- free call free twice lead code injection virus worms form program threat virus avirus fragment code embed- ded legitimate program virus self replicate design infect program wreak havoc system modify destroy file cause system crash program malfunction penetration attack direct attack system virus specific architecture operate system application virus par- ticular problem user pc unix multiuser operate system generally susceptible virus executable program protect write operate system virus infect program power usually limited aspect system protect virus usually bear spam e mail phishing attack spread user download viral program internet file share service exchange infected disk adistinction virus require human activity worm use network replicate help human example virus infect host consider microsoft office file file contain macro visual basic program program office suite word powerpoint excel execute automatically program run user account macro run largely unconstrained example delete user file follow code sample show simple write visual basic macro worm use format hard drive windows computer soon file contain macro open set ofs = createobject(”scripting filesystemobject vs = shell(”c command.com /k format c:”,vbhide commonly worm e mail user contact list virus work virus reach target machine program know virus dropper insert virus system virus dropper usually trojan horse execute reason instal virus core activity instal virus number thing literally thousand virus fall main category note virus belong category file standard file virus infect system append file change start program execution jump code execute return

append file change start program execution jump code execute return control program execution notice file virus know parasitic virus leave file leave host program functional boot boot virus infect boot sector system execute time system boot operate system load watch bootable medium infect virus know memory virus appear file system figure 16.5 show boot virus work boot virus adapt infect firmware network card pxe extensible firmware interface efi environment macro virus write low level language assembly c. macro virus write high level language visual basic virus trigger program capable execute macro run example macro virus contain rootkit originally coin describe door unix system mean provide easy root access term expand virus malware infiltrate operate system result complete system compromise aspect system deem trust malware infect operate system system function include function normally facilitate removable r w disk instal infect logic bomb wreak havoc original boot block system boot virus memory hide memory new limit virus attach disk read- write interrupt monitor block attempt program write virus copy boot sector unused boot sector computer virus source code source code virus look source code modify include virus help spread virus polymorphic polymorphic virus change time instal avoid detection antivirus software change affect virus functionality change virus signature virus signature pattern identify virus typically series byte virus code encrypt encrypt virus include decryption code encrypt virus avoid detection virus decrypt stealth tricky virus attempt avoid detection modify part system detect example modify read system file modify read original form code return infect code multipartite avirus type able infect multiple part system include boot sector memory file make difficult detect armored armored virus obfuscate write hard antivirus researcher unravel understand com- press avoid detection disinfection addition virus dropper file virus infestation frequently hide file attribute unviewable file name vast variety virus continue grow example

attribute unviewable file name vast variety virus continue grow example 2004 widespread virus detect exploit separate bug oper- ation virus start infect hundred windows server includ- e trusted site run microsoft internet information server iis vulnerable microsoft explorer web browser visit site receive browser virus download browser virus instal door program include keystroke logger record enter keyboard include password credit card number instal daemon allow unlimited remote access intruder allow intruder route spam infected desktop active security relate debate computing community con- cern existence monoculture system run hardware operating system application software monoculture sup- posedly consist microsoft product question mono- culture exist today question increase threat damage cause virus security intrusion vul- nerability information buy sell place like dark web world wide web system reachable unusual client configuration method system attack affect valuable attack 16.3 system network threats program threat pose security risk risk compound order magnitude system connect network worldwide connectivity make system vulnerable worldwide attack open operate system service enable function allow likely bug available exploit increasingly operate system strive secure default example solaris 10 move model service ftp telnet enable default system instal model service disable installation time specifically enable system administrator change reduce system attack surface hacker leave track network traffic pat- tern unusual packet type mean reason hacker frequently launch attack zombie system independent system device compromise hacker continue serve own- er owner knowledge nefarious purpose system network threats standard security attacks.1

include denial service attack spam relay zombie hacker par- ticularly difficult track mask original source attack identity attacker reason secure incon- sequential system system contain valuable information service lest turn stronghold hacker widespread use broadband wifi exacerbate difficulty track attacker simple desktop machine easily compromise malware valuable machine bandwidth network access wireless ethernet make easy attacker launch attack join public network anonymously wardriving"—locate private unprotected network target attack network traffic networks common attractive target hacker option mount network attack show figure 16.6 attacker opt remain passive intercept network traffic attack commonly refer sniffin obtain useful information type session conduct system session content alternatively attacker active role masquerade party refer spoofin fully active man middle intercept possibly modify transaction peer describe common type network attack denial service dos attack note possible guard attack mean encryption authentication discuss later chap- ter internet protocol support encryption authenti- cation default denial service mention early denial service attack aim gain infor- mation steal resource disrupt legitimate use sys- tem facility attack involve target system facility attacker penetrate launch attack prevent legitimate use frequently easy break system facility denial service attack generally network base fall category attack category use facility resource essence useful work example website click download java applet proceed use available cpu time pop window infinitely second category involve disrupt network facility successful denial service attack kind major website attack hour day cause partial failure attempt use target facility attack usually stop network level operate system update reduce vulnerability generally impossible prevent denial service attack attack use mechanism normal operation difficult prevent resolve distribute denial service ddos attack attack launch multiple site common target typically zombie ddos attack common asso- ciate blackmail attempt site come attack attacker offer halt attack exchange money site know attack difficult determine system slowdown attack surge system use consider

attack difficult determine system slowdown attack surge system use consider successful advertising campaign greatly increase traffic site consider ddos.

interesting aspect dos attack example authentication algorithm lock account period time incorrect attempt access account attacker cause authentication block purposely make incorrect attempt access account similarly firewall automatically block certain kind traf- fic induce block traffic example suggest programmer system manager need fully understand algorithm technology deploy finally computer science class notorious source accidental system dos attack consider programming exercise student learn create subprocesse thread common bug involve spawn subprocesse infinitely system free memory cpu resource stand chance cryptography security tool port scanning attack means hacker detect system vulnerability attack security personnel use port scanning example detect service need suppose run port scanning typically automate involve tool attempt create tcp ipconnection send udp packet specific port range port scanning reconnaissance technique know fin- gerprinting attacker attempt deduce type operate system use set service order identify know vulnerability server client easy disclose exact version number network protocol header example http server user agent header detailed analysis idiosyncratic behavior protocol handler help attacker figure operating system target necessary step successful exploitation network vulnerability scanner sell commercial product tool perform subset functionality scanner example nmap http://www.insecure.org/nmap/ versatile open- source utility network exploration security auditing point target determine service run include application name version identify host operating system provide information defense firewall defend target exploit know bug tool metasploit pick port scanner leave provide payload construction facility test vulnerability exploit create specific payload trigger bug seminal work port scanning technique find http://phrack.org/issues/49/15.html technique constantly evolve measure detect form basis network intrusion detection system discuss later 16.4 cryptography security tool defense computer attack run gamut methodology technology broad tool available system designer user cryptography section discuss cryptography use computer security note cryptography discuss simplify educational purpose reader caution scheme describe real world good cryptography library widely available good basis production application isolated computer operate system

widely available good basis production application isolated computer operate system reliably determine sender recipient interprocess communication control communication channel computer network computer situation different networked computer receive bit wire immediate reliable way determine machine application send bit similarly computer send bit network way know eventually receive additionally send receive system way know eavesdropper listen communication commonly network address infer potential sender receiver network message network packet arrive source address ip address computer send message name intend receiver specify destination address appli- cation security matter ask trouble assume source destination address packet reliably determine send receive packet rogue computer send message falsified source address numerous computer specify destination address typically receive packet example router way destination receive packet operate system decide grant request trust name source request suppose provide protection request datum determine receive response message content send network generally consider infeasible build network scale source destination address packet trust sense alternative eliminate need trust network job cryptography abstractly cryptography constrain potential sender and/or receiver message modern cryptography base secret call key selectively distribute computer network process message cryptography enable recipient message verify message create computer possess certain key similarly sender encode message computer certain key decode message unlike network address key design computationally feasible derive message generate public information provide trustworthy mean constrain sender receiver message cryptography powerful tool use cryptography cause contention country ban use certain form limit long key ongoing debate technology vendor smartphone vendor provide door include cryp- tography allow law enforcement bypass privacy provide observer argue door intentional security weakness exploit attacker misuse government finally note cryptography field study unto large small complexity subtlety explore important aspect part cryptography pertain operate system solve wide variety communication security problem encryp- tion frequently aspect modern computing send message securely network protect database datum file entire disk have content

securely network protect database datum file entire disk have content read unauthorized entity encryption algorithm enable sender message ensure computer possess certain key read message ensure writer datum reader datum encryption message ancient practice course encryption algorithm cryptography security tool date ancient time section describe important modern encryption principle algorithm encryption algorithm consist following component set k key set m message set c ciphertext encrypting function e k (m c k k ek function generate ciphertext message e ek k efficiently computable function generally ek randomized mapping message ciphertext decrypting function d k (c m k k dk function generate message ciphertext d dk k efficiently computable function encryption algorithm provide essential property give ciphertext c c computer compute m ek(m = c possess k.

computer hold k decrypt ciphertext plaintext produce computer hold k decrypt ciphertext

ciphertext generally expose example send network important infeasible derive k ciphertext main type encryption algorithm symmetric asym- metric discuss type follow section symmetric encryption algorithm key encrypt decrypt secrecy k protect figure 16.7 show example user communicate securely symmetric encryption insecure channel note key exchange place directly party trust party certificate author- ity discuss section 16.4.1.4 past decade commonly symmetric encryp- tion algorithm united states civilian application data- encryption standard des cipher adopt national institute stan- dards technology nist des work take 64 bit value 56 bit key perform series transformation base substitution permutation operation des work block bit time know block cipher transformation typical block cipher block cipher key encrypt extended datum vulnerable attack des consider insecure application key exhaustively search moderate compute resource note frequently give des nist create modification call triple des des algorithm repeat time encryption decryption plaintext key example c = ek3(dk2(ek1(m key effective key length 168 bit c = ek(m m = dk(c secure communication insecure medium.2 2001 nist adopt new block cipher call advanced encryption standard aes replace des aes know rijndael standard- ize fips-197 http://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf

use key length 128 192 256 bit work 128 bit block gen- erally algorithm compact efficient block cipher necessarily secure encryption scheme particular directly handle message long require block size alternative stream cipher securely encrypt long stream cipher design encrypt decrypt stream byte bit block useful length communication block cipher slow key input pseudo random- bit generator algorithm attempt produce random bit output generator feed key keystream keystream infinite set bit encrypt plaintext stream xor operation xor exclusive operation compare input bit generate output bit bit result 0 bit different result 1 aes base cipher suite include stream cipher common today cryptography security tool asymmetric encryption algorithm different encryption decryption key entity prepare receive encrypt communication create key make call public key available want sender use key encrypt communication key creator decrypt communication scheme know public key encryption breakthrough cryptography describe diffie hellman https://www-ee.stanford.edu/ man publications/24.pdf long key keep secret deliver securely instead encrypt message receive entity matter listen entity decrypt message example public key encryption work describe algo- rithm know rsa inventor rivest shamir adleman rsa widely asymmetric encryption algorithm asymmetric algo- rithm base elliptic curve gain ground key length algorithm short crypto- rsa ke public key kd private key n product large randomly choose prime number p q example p q 2048 bit computationally infeasible derive kd n ke n ke need keep secret widely disseminate encryption algorithm eke n(m = mke mod n ke satisfy kekd mod p 1)(q 1 = 1 decryption algorithm dkd n(c = ckd mod n. example small value show figure 16.8 example p = 7 q = 13 calculate n = 7 13 = 91 p1)(q1 = 72

select ke relatively prime 72 < 72 yield 5 finally calculate kd kekd mod 72 = 1 yield 29 key public key ke n = 5 91 private key kd n = 29 91 encrypt message 69 public key result message 62 decode receiver private key use asymmetric encryption begin publication public key destination bidirectional communication source publish public key publication simple hand electronic copy key complex private key secret key zealously guard hold key decrypt message create match public key note seemingly small difference key use asymmetric symmetric cryptography large practice asymmet- ric cryptography computationally expensive execute fast computer encode decode ciphertext usual symmetric algorithm asymmetric algorithm use asymmetric algorithm truth algorithm general- purpose encryption large amount datum encryption small amount datum authentication confiden- tiality key distribution follow section see encryption offer way constrain set possible receiver message constrain set potential sender message 695 mod 91 6229 mod 91 encryption decryption rsa asymmetric cryptography.3

call authentication authentication complementary encryption authentication useful prove message modify discuss authentication constraint possible sender mes- sage note sort authentication similar distinct user authentication discuss section 16.5 authentication algorithm symmetric key consist follow- set k key set m message set authenticator function s k (m a k k sk function generate authenticator message s sk k efficiently computable function cryptography security tool function v k (m × {true false k k vk function verify authenticator message v vk k efficiently computable function critical property authentication algorithm possess message m computer generate authenticator a vk(m = true possess k. computer hold k generate authenticator message computer possess k verify computer hold k generate authenticator mes- sage verify vk authenticator generally expose example send network message feasible derive k authenticator practically vk(m = true know m modify sender message k. share k entity know message originate type encryption algorithm main variety authentication algorithm step understand algorithm explore hash function hash function h(m create small fixed sized block datum know message digest hash value message m. hash function work take message split block process block produce n bit hash h collision resis- tant infeasible find m′ m h(m = h(m′ h(m = h(m′ know m = m′—that know message modify common message digest function include md5 consider insecure produce 128 bit hash sha-1 output 160 bit hash message digest useful detect change message useful authenticator example h(m send message h know modify m m′ recompute h(m′ message modification detect authenticate h(m main type authentication algorithm use symmetric encryption message authentication code mac cryptographic checksum gener- ate message secret key mac provide way securely authenticate short value use authenticate h(m h collision resistant obtain way securely authenticate long message hash note k need compute sk vk able compute compute second main type authentication algorithm

sk vk able compute compute second main type authentication algorithm digital signature algorithm authenticator produce call digital signature digital signature useful enable verify authenticity message digital signature algorithm computation- ally infeasible derive ks kv kv public key ks private consider example rsa digital signature algorithm similar rsa encryption algorithm key use reverse digital signature message derive compute sks(m = h(m)ks mod n. key ks pair d n n product large randomly choose prime number p q. verification algorithm vkv(m = akv mod n = h(m kv satisfy kvks mod p 1)(q 1 = 1 digital signature case aspect cryptography entity message example creator program sign code digital signature validate code modify publication installation computer code signing common security improvement method system note encryption authentication sepa- rately instance want authentication confidentiality example company provide software patch sign patch prove come company modify authentication component aspect security example digital signature core nonrepudiation supply proof entity perform action typical example nonrepudiation involve filling electronic form alternative signing paper contract nonrepudiation assure person fill electronic form deny certainly good battle cryptographer invent cipher cryptanalyst try break involve key symmetric algorithm party need key delivery symmetric key huge challenge perform band example walter want communicate rebecca securely exchange key paper document conver- sation communication electronically method scale consider key management challenge suppose lucy want communicate n user privately lucy need n key security need change key frequently reason effort create asymmetric key algorithm key exchange public give user audra need private key matter people want communicate matter manage public key recipient communication public key need secure simple storage key ring unfortunately distribution public key require care consider man middle attack show figure 16.9 person want receive encrypt message send public key attacker send bad public key match private key person want send encrypt message know

key match private key person want send encrypt message know well use bad key encrypt message attacker happily decrypt problem authentication need proof own public key way solve problem involve use digital certificate digital certificat public key digitally sign trust party trust party receive proof identification entity certify public key belong entity know trust certifier certificat authority public key include web browser consumer certificate distribute certificate authority vouch authority digitally sign public key authority create web trust certificate distribute standard cryptography security tool man middle attack asymmetric cryptography.4 x.509 digital certificate format parse computer scheme secure web communication discuss section 16.4.3 implementation cryptography network protocol typically organize layer layer act client protocol generate message send protocol peer machine hand message protocol network protocol stack delivery peer machine example ip network tcp transport layer protocol act client ip network layer protocol tcp packet pass ip delivery ip peer end connection ip encapsulate tcp packet ip packet similarly pass data link layer transmit network peer destination computer ip peer deliver tcp packet tcp peer machine seven layer include osi model mention early describe detail section 19.3.2 cryptography insert layer network protocol stack tls section 16.4.3 example provide security transport layer network layer security generally standardize ipsec define ip packet format allow insertion authenticator encryption packet content ipsec use symmetric encryption use internet key exchange ike protocol key exchange ike base public- key encryption ipsec widely basis virtual private network vpns traffic ipsec endpoint encrypt private network public numerous proto- col develop use application pgp encrypt- ing e mail type scheme application code implement security cryptographic protection well place protocol stack gen- eral definitive answer hand protocol benefit protection place lower stack example ip packet encapsu- late tcp packet encryption ip packet ipsec example hide content encapsulate tcp packet similarly authenticator ip packet

example hide content encapsulate tcp packet similarly authenticator ip packet detect modification contain tcp header information hand protection low layer protocol stack insufficient protection high layer protocol example appli- cation server accept connection encrypt ipsec able authenticate client computer request receive authenticate user client computer server need use application level protocol user require type password consider problem e mail e mail deliver industry standard smtp protocol store forward frequently multiple time deliver transmission secure insecure network e mail secure e mail message need encrypt security independent transport carry unfortunately like tool encryption good evil ransomware attack describe early example base encryption mention attacker encrypt information target system render inaccessible owner idea force owner pay ransom key need decrypt datum prevention attack take form well system network security well- execute backup plan content file restore example tls transport layer security tls cryptographic protocol enable computer communicate securely limit sender receiver message commonly cryptographic protocol internet today standard protocol web browser communicate securely web server completeness note tls evolve ssl secure sock- ets layer design netscape describe detail cryptography security tool tls complex protocol option present single variation describe simplified abstract form maintain focus use cryptographic primitive complex dance asymmetric cryptography client server establish secure session key symmetric encryption session avoid- e man middle replay attack add cryptographic strength session key forget session complete communi- cation

require generation new session key tls protocol initiate client c communicate securely server prior protocol use server s assume obtain certificate denote cert certification authority certificate structure contain follow attribute attr server unique distinguished common dns identity asymmetric encryption algorithm e server public key ke server validity interval interval certificate consid- digital signature information = skca(attrs eke interval addition prior protocol use client presume obtain public verification algorithm vkca case web user browser ship vendor contain verification algorithm public key certain certification authority user delete add c connect s send 28 byte random value nc server respond random value n plus certificate cert client verify vkca(attrs eke interval = true current time validity interval interval test satisfy server prove identity client generate random 46 byte premaster secret pm send cpm = eke(pms server server recover pm = dkd(cpms client server possession nc n pm compute share 48 byte master secret ms = h(nc ns pms server client compute m know pm dependence m nc ns ensure ms fresh value session key previous communication point client server compute following key symmetric encryption key k encrypt message client server symmetric encryption key k encrypt message server client mac generation key k generate authenticator message client server mac generation key k generate authenticator message server client send message m server client send c = ek cs m sk receive c server recover m a= dk accept m vk cs m = true similarly send message m client server send c = ek sc m sk client recover

m a= dk accept m vk sc m = true protocol enable server limit recipient message client generate pm limit sender message accept client similarly client limit recipient message send sender message accept party know kd party decrypt cpm application web transaction client need verify identity party know kd purpose certificate cert particular attr field contain information client use determine identity example domain server communicate application server need information client tls support option client send certificate server addition use internet tls wide variety task example mention early ipsec widely basis virtual private network vpn ipsec vpns competitor tls vpn ipsec good point point encryption traffic company office tls vpns flexible efficient individual employee work remotely corporate 16.5 user authentication early discussion authentication involve message session user system authenticate user authenticate message come user pointless major security problem operate system user authentication protection system depend ability identify program process currently execute turn depend ability identify user system user normally identify determine user identity authentic generally user authentication base thing user possession key card user knowledge user identifier password attribute user fingerprint retina pattern signature common approach authenticate user identity use password user identify user id account ask password user supply password match password store system system assume account access owner account password protect object computer system absence complete protection scheme consider special case key capability instance password associate resource file request use resource password give password correct access grant different password associate different access right example different password read file append file update file practice system require password user gain right password theoretically secure system tend implement classic trade security convenience security make inconvenient security frequently bypass circumvent password extremely common easy understand use unfortunately password guess accidentally expose sniff read eavesdropper illegally transfer authorized user unauthorized common way guess password

eavesdropper illegally transfer authorized user unauthorized common way guess password way intruder human program know user information user frequently people use obvious information name cat spouse password way use brute force try enumeration possible combination valid password character letter number punctuation systems)—until password find short password especially vulnerable method example character password provide 10,000 variation average guess 5,000 time produce correct hit program try password millisecond 5 second guess character password enumeration successful system allow long password include uppercase lowercase letter number punctuation character course user advantage large password space example use lowercase letter common method dictionary attack word word variation common password try addition guess password expose result visual electronic monitoring intruder look shoulder user shoulder surfin user log learn password easily watch keyboard alternatively access network computer reside seamlessly add network monitor allow sniff watch datum transfer network include user id password encrypt data stream contain password solve problem system password steal example file contain password copy system analysis consider trojan horse program instal system capture keystroke send application common method grab password specially debit card passcode instal physical device code record user example skimmer atm machine device instal keyboard computer exposure particularly severe problem password write read lose system force user select hard to- remember long password change password frequently cause user record password reuse result sys- tem provide security system allow user select easy final type password compromise illegal transfer result human nature computer installation rule forbid user share account rule implement accounting reason aim improve security instance suppose user id share user security breach occur user id impossible know id time break occur user authorized user user id user question directly use account addition user notice different account detect break user break account share rule help friend circum- vent accounting behavior result system access unauthorized

help friend circum- vent accounting behavior result system access unauthorized user possibly harmful one password generate system select user system generate password difficult remember user write mention user select password easy guess user favorite car example system check propose password ease guessing crack accept system age password force user change password regular interval month instance method foolproof user easily toggle password solution implement system record password history user instance system record n password allow reuse variant simple password scheme exam- ple password change frequently extreme pass- word change session session new password select system user end session password session case password unauthorized person person use legitimate user try use invalid password session discover security violation step take repair breach security problem approach difficulty keep pass- word secret computer system store password securely allow use authentication user present pass- strong easy remember password extremely important use strong hard guess hard shoulder surf password critical system like bank account important use password lot system important easily hack system reveal password use important system good technique generate password letter word easily remember phrase upper lower character number punctuation mark throw good measure example phrase girlfriend katherine yield password mgn.isk password hard crack easy user remember secure system allow character password system allow password include space character user create passphrase easy remember difficult break word unix system use secure hashing avoid necessity keep password list secret password hash encrypt impossible system decrypt store value determine hash function easy compute hard impossible invert give value x easy compute hash function value f(x give function value f(x impossible compute x. function encode password encode password store user present password hash compare store encode password store encode password see decode password determine password file need keep secret drawback method system long control password password hash copy password

drawback method system long control password password hash copy password file run fast hash routine hash word dictionary instance compare result password user select password word dictionary password crack sufficiently fast computer cluster slow computer comparison hour furthermore system use know hashing algorithm attacker cache password crack previously reason system include salt record random number hashing algorithm salt value add password ensure plaintext password result different hash value addition salt value make hash dictionary ineffective dictionary term need combine salt value comparison store password new version unix store hashed password entry file readable superuser program compare hash store value run setuid root read file user avoid problem password sniffing shoulder surfing system use set pair password session begin system randomly select present password pair user supply system user challenge respond correct answer challenge approach generalize use algorithm password scheme system user share symmetric password password pw transmit medium allow exposure password input function challenge ch present system user compute function h(pw ch result function transmit authenticator computer computer know pw ch perform computation result match user authenticate time user need authenticate ch generate step ensue time authenticator different algorithmic password susceptible reuse user type password entity intercept password able reuse time password system way prevent improper authentication password exposure time password system implement way commer- cial implementation use hardware calculator display display numeric keypad calculator generally shape credit card key chain dongle usb device software run computer smartphone provide user h(pw ch pw input user generate calculator synchronization computer some- time pw personal identificatio number pin output system show time password time password gener- ator require input user involve factor authentication different type component need case example one- time password generator generate correct response pin valid factor authentication offer far well authentication protection single factor authentication require know variation use

well authentication protection single factor authentication require know variation use password authentication involve use biometric measure palm- hand reader commonly secure physical access example access datum center reader match store parameter read hand reader pad parameter include temperature map finger length finger width line pattern device currently large expensive normal computer authentication fingerprint reader accurate cost effective device read finger ridge pattern convert sequence number time store set sequence adjust location finger read pad factor software scan finger pad compare feature store sequence determine match course multiple user profile store scanner differentiate accurate factor authentication scheme implement security defenses result require password user fingerprint scan information encrypt transit system resistant spoofing replay attack multifactor authentication well consider strong authentica- tion usb device plug system pin fingerprint scan have place finger pad plug usb system authentication method convenient normal password recall strong authentication sufficient guarantee id user authenticate session hijack encrypt 16.6 implement security defenses myriad threat system network security security solution solution range improve user education technology write well software security professional subscribe theory defense depth state layer defense well few layer course theory apply kind security consider security house door lock door lock lock alarm section look major method tool technique improve resistance threat note security improve technique properly protection security cover chapter 17 step improve security aspect computing security policy policy vary widely generally include statement secure example

policy state outside- accessible application code review deploy user share password connection point company outside port scan run month policy place impossible user administrator know permissible require allow policy road map security site try secure secure need map know security policy place people affect know guide policy live document review update periodically ensure pertinent determine security policy correctly imple- mente good way execute vulnerability assessment assess- ment cover broad ground social engineering risk assess- ment port scan risk assessment example attempt value asset entity question program management team system facil- ity determine odd security incident affect entity decrease value odd suffer loss potential loss know value place try secure entity core activity vulnerability assessment penetration test entity scan know vulnerability book concern operate system software run concentrate aspect vulnerability assessment vulnerability scan typically time computer use rela- tively low minimize impact appropriate test system production system induce unhappy behavior target system network device scan individual system check variety aspect short easy guess password unauthorized privileged program setuid program unauthorized program system directory unexpectedly long run process improper directory protection user system directory improper protection system datum file password file device file operating system kernel dangerous entry program search path example trojan horse discuss section 16.2.1 current directory easily write directory /tmp change system program detect checksum value unexpected hidden network daemon problem find security scan fix automatically report manager system networked computer susceptible security attack standalone system attack know set access point directly connect terminal face attack unknown large set access point potentially severe security problem less extent system connect telephone line modem fact u.s. government consider system secure far reach connection instance secret system access building consider secret system lose secret rating form communication occur outside environment government facility extreme security precaution connector plug terminal secure computer lock safe

extreme security precaution connector plug terminal secure computer lock safe office terminal use person proper id gain access building office know physical lock com- bination know authentication information computer gain access computer example multifactor authentication unfortunately system administrator computer security profes- sional frequently impossible lock machine room disallow implementing security defenses remote access instance internet currently connect billion com- puter device mission critical indispensable resource company individual consider internet club club million member good member bad member bad member tool use attempt gain access interconnected computer vulnerability scan apply network address problem network security scan search network port respond request service enable access block disable scan determine detail application listen port try determine know vulnerability test vulnerability determine system misconfigure lack need patch finally consider use port scanner hand attacker try improve security tool help attack- er find vulnerability attack fortunately possible detect port scan anomaly detection discuss general challenge security tool good harm fact people advocate security obscurity state tool write test security tool find exploit security hole believe approach security valid point example attacker write tool reasonable security obscurity consider layer security long layer example company publish entire network configuration keep information secret make hard intruder know attack company assume information remain secret false sense secure system facility intimately link intrusion detection prevention intrusion prevention suggest strive detect attempt successful intrusion computer system initiate appro- priate response intrusion intrusion prevention encompass wide array technique vary number axis include follow time detection occur detection occur real time intrusion occur fact type input examine detect intrusive activity include user shell command process system call network packet header content form intrusion detect correlate information source range response capability simple form response include alert- e administrator potential intrusion halt potentially intrusive activity example kill process engage activity sophisticated form response system transpar-

example kill process engage activity sophisticated form response system transpar- ently divert intruder activity honeypot false resource expose attacker resource appear real attacker enable system monitor gain information attack degree freedom design space detect intrusion yield wide range solution know intrusion prevention system ips ipss act self modify firewall pass traffic intrusion detect point traffic block constitute intrusion define suitable specification intrusion turn difficult automatic ips today typically settle ambitious approach call signature base detection system input network traffic examine specific behavior pattern signature know indicate attack simple example signature base detection scan network packet string /etc passwd target unix system example virus- detection software scan binary network packet know virus second approach typically call anomaly detection attempt technique detect anomalous behavior computer system course anomalous system activity indicate intrusion presumption intrusion induce anomalous behavior example anomaly detection monitor system call daemon process detect system behavior deviate normal pattern possibly indicate buffer overflow exploit daemon corrupt behavior example monitor shell command detect anomalous command give user detect anomalous login time user indicate attacker succeed gain access user account signature base detection anomaly detection view side coin signature base detection attempt characterize dan- gerous behavior detect behavior occur anomaly detection attempt characterize normal nondangerous behav- ior detect behavior occur different approach yield ips different property how- particular anomaly detection find previously unknown method intrusion call zero day attack signature base detection contrast identify know attack codify recognizable pat- tern new attack contemplate signature generate evade signature base detection problem know vendor virus detection software release new signature great frequency new virus detect manually anomaly detection necessarily superior signature base detection significant challenge system attempt anomaly detection benchmark normal system behavior accurately sys- tem penetrate benchmarke intrusive activity include normal benchmark system benchmarke cleanly influence intrusive behavior bench- mark fairly complete picture

benchmarke cleanly influence intrusive behavior bench- mark fairly complete picture normal behavior number false positive false alarm worse false negative miss intrusion excessive illustrate impact marginally high rate false alarm consider installation consist unix workstation implement security defenses security relevant event record purpose intrusion detection small installation easily generate million audit record day worthy administrator investigation suppose optimistically actual attack reflect audit record roughly compute rate occurrence audit record reflect truly intrusive activity follow interpret probability occurrence intrusive record denote p(i event occurrence record reflect truly intrusive behavior p(i = 0.00002 know p(¬i = 1 p(i = 0.99998 let denote raising alarm ids accurate ids maximize p(i|a p(¬i|¬a)—that probability alarm indicate intrusion alarm indicate intrusion focus p(i|a moment compute bayes theorem p(i p(a|i + p(¬i p(a|¬i 0.00002 p(a|i + 0.99998 p(a|¬i consider impact false alarm rate p(a|¬i p(i|a good true alarm rate p(a|i = 0.8 seemingly good false- alarm rate p(a|¬i = 0.0001 yield p(i|a 0.14 few seven alarm indicate real intrusion system security administrator investigate alarm high rate false alarm call christmas tree effect”—is exceedingly wasteful quickly teach administrator ignore alarm example illustrate general principle ips usability offer extremely low false alarm rate achieve sufficiently low false- alarm rate especially challenge anomaly detection system mention difficulty adequately benchmarke normal system behavior research continue improve anomaly detection technique intrusion detection software evolve implement signature anomaly algorithm algorithm combine result arrive accurate anomaly detection rate see virus wreak havoc system protection virus important security concern antivirus program provide protection program effective particular know virus work search program system specific pattern instruction know virus find know pattern remove instruction disinfect program antivirus program catalog thousand virus search virus antivirus software continue sophisti- cat virus modify infect software avoid basic pattern match approach antivirus program antivirus program turn look family pattern single pattern

antivirus program antivirus program turn look family pattern single pattern iden- tify virus fact antivirus program implement variety detection algorithm decompress compress virus check sig- nature look process anomaly process open executable file write suspicious example compiler pop- ular technique run program sandbox section 17.11.3 controlled emulate section system antivirus software analyze behavior code sandbox let run unmonitored antivirus program complete shield scan file file system search boot sector memory inbound outbound e mail file download file removable device medium good protection computer virus prevention prac- tice safe computing purchase unopened software vendor avoid free pirate copy public source disk exchange offer safe route prevent infection new copy legitimate software application immune virus infection case dis- gruntled employee software company infect master copy software program economic harm company likewise hard- ware device come factory pre infected convenience macro virus defense exchange microsoft word document alternative file format call rich text format rtf unlike native word format rtf include capability attach macro defense avoid open e mail attachment unknown user unfortunately history show e mail vulnerability appear fast fix example 2000 love bug virus widespread travel e mail message pretend love note send friend receiver receiver open attach visual basic script virus propagate send address receiver e mail contact list fortunately clog e mail system user inboxe relatively harmless effectively negate defensive strategy open attachment people know receiver effective defense method avoid open e mail attachment contain executable code company enforce policy remove incoming attachment e mail message safeguard prevent infection permit early detection user begin completely reformatte hard disk especially boot sector target viral attack secure software upload signature program take secure message digest computation result file associate message- digest list keep free unauthorized access periodically time program run operate system recompute signature implementing security defenses compare signature original list

system recompute signature implementing security defenses compare signature original list difference serve warning possible infection technique combine example high overhead antivirus scan sandbox program pass test signature create signature match time program run need virus scan auditing accounting logging auditing accounting logging decrease system performance useful area include security logging general spe- cific system execution log analysis program behavior misbehavior typically suspicious event log authentica- tion failure authorization failure tell lot break accounting potential tool security administrator kit find performance change turn reveal security problem early unix computer break in detect cliff stoll examine accounting log spot anomaly firewalling protect system network turn question trusted computer connect safely untrustworthy network solution use firewall separate trusted untrusted system firewal computer appliance process router sit trusted untrusted anetwork firewall limit network access multiple security domain monitor log connection limit connection base source destination address source destination port direction connection instance web server use http communicate web browser firewall allow http pass host outside firewall web server firewall worm morris internet worm finger protocol break computer finger allow pass example fact network firewall separate network multiple domain common implementation internet untrusted domain semitrusted semisecure network call demilitarized zone dmz domain company computer domain figure 16.10 connection allow internet dmz computer company computer internet allow internet dmz computer company computer optionally control communication allow dmz company computer instance web server dmz need query database server corporate network firewall access contain dmz system break unable access company computer course firewall secure attack proof ability secure connection compromise furthermore firewall prevent attack tunnel travel protocol connection internet access company dmz access internet access dmz domain separation firewall firewall allow buffer overflow attack web server stop firewall example http connection allow content http connection house attack likewise denial- service attack affect firewall machine vulnerability firewall spoofing unauthorized

denial- service attack affect firewall machine vulnerability firewall spoofing unauthorized host pretend authorized host meet authorization criterion example firewall rule allow connection host identify host ip address host send packet address allow firewall addition common network firewall new kind firewall pro con apersonal firewal software layer include operate system add application limit communication security domain limit com- munication possibly give host user add personal firewall pc trojan horse deny access net- work pc connect example application proxy fire wall understand protocol application speak network example smtp mail transfer application proxy accept connec- tion smtp server initiate connection original destination smtp server monitor traffic forward message watch disable illegal command attempt exploit bug firewall design specific protocol xml firewal example specific purpose analyze xml traffic block disallowed malformed xml system firewall sit application kernel monitor system execution example solaris 10 privilege feature implement list system call process allow process need spawn process ability take away instance implement security defense ongoing battle cpu designer operating system implementer hacker particular technique helpful defend code injection mount code injection attack hacker able deduce exact address memory target normally difficult memory layout tend predictable operating system technique call address space layout randomization aslr attempt solve problem randomize address space put address space start location stack heap unpredictable location address randomization foolproof make exploitation consid- erably difficult aslr standard feature operate system include windows linux macos mobile operate system ios android approach adopt place user datum system file separate parti- tion system partition mount read datum partition read write approach numerous advantage great security system partition file easily tamper bolster system integrity android take step linux dm verity mechanism cryptographically hash system partition detect modification security defense summarize apply appropriate layer defense system safe persistent attacker summary layer include educate user safe computing don’t attach device

summary layer include educate user safe computing don’t attach device unknown origin computer share password use strong password avoid fall social engineering appeal realize e mail necessarily private communication educate user prevent phishing attack don’t click e-mail attachment link unknown known sender authen- ticate example phone request legitimate use secure communication possible physically protect computer hardware configure operate system minimize attack surface disable configure system daemon privilege application service secure possible use modern hardware software likely date system application date patch run application trusted source code enable logging auditing review log periodically automate install use antivirus software system susceptible virus software date use strong password passphrase record find use intrusion detection firewalling network base protection system appropriate important facility use periodic vulnerability assessment testing method test security response incident encrypt mass storage device consider encrypt important individ- ual file security policy important system facility 16.7 example windows 10 microsoft windows 10 general purpose operate system design sup- port variety security feature method section examine feature windows 10 use perform security function infor- mation background windows appendix b. windows 10 security model base notion user account windows 10 allow creation number user account group manner access system object permit deny desire user identify system unique security id user log windows 10 create security access token include security id user security id group user member list special privilege user example special privilege include back file directory shut computer log interactively change system clock process windows 10 run behalf user receive copy access token system use security id access token permit deny access system object user process behalf user attempt access object authentication user account typically accomplish user password modular design windows 10 allow development custom authentication package example retinal eye scanner verify user say windows 10 use idea subject ensure program

verify user say windows 10 use idea subject ensure program run user great access system user authorize subject track manage permission program user run compose user access token program act behalf user windows 10 operate client server model class subject control access simple subject server subject example simple subject typical application program user execute log simple subject assign security example windows 10 context base security access token user server subject process implement protect server use security context client act client behalf mention section 16.6.6 auditing useful security technique windows 10 build auditing allow common security threat monitor example include failure auditing login logoff event detect random password break in success auditing login logoff event detect login activity strange hour success failure write access auditing executable file track virus outbreak success failure auditing file access detect access sensitive file windows vista add mandatory integrity control work assign- e integrity label securable object subject order give subject access object access request dis- cretionary access control list integrity label equal high secure object give operation integrity label windows 7 untrusted low medium high system addition access mask bit permit integrity label noreadup nowriteup noexecuteup nowriteup automatically enforce low integrity subject perform write operation high integrity object how- explicitly block security descriptor perform read execute operation securable object explicit integrity label default label medium assign label give subject assign logon instance nonadministrative user integrity label medium addition integrity label windows vista add user account control uac represent administrative account build admin- istrator account separate token normal usage build administrators group disable integrity label medium elevated usage build administrators group enable integrity label high security attribute object windows 10 describe security descriptor security descriptor contain security id owner object change access permission group security id posix subsystem discretionary access control list identify user group allow

posix subsystem discretionary access control list identify user group allow explicitly deny access system access control list control auditing message system generate optionally system access control list set integrity object identify operation block low integrity subject read write enforce execute example security descriptor file foo.bar owner gwen discretionary access-owner gwen access group cs read write access user maddie access addition system access control list tell system audit write integrity label medium deny read write execute low integrity subject access control list compose access control entry contain security id individual group grant access access mask define possible action object value accessallowe accessdenie action file windows 10 following access type readdata writedata appenddata execute writeattributes allow fine degree control access object windows 10 classify object container object noncontainer object container object directory logically contain object default object create container object new object inherit permission parent object similarly user copy file directory new directory file inherit permission destination directory noncontainer object inherit permission furthermore permission change directory new permission automatically apply existing file subdirectory user explicitly apply desire system administrator use windows 10 performance monitor help spot approach problem general windows 10 good job provide feature help ensure secure computing environment feature enable default reason myriad security breach windows 10 system reason vast number service windows 10 start system boot time number application typically instal windows 10 system real multiuser environment system administrator formulate security plan implement feature windows 10 provide feature differentiate security windows 10 early version code signing version windows 10 mandatory application properly sign author execute version optional leave administrator determine unsigned application protection internal problem security contrast consider computer system environment people building busi- ness valuable object threat system datum store computer system protect unautho- rize access malicious destruction alteration accidental introduc- tion inconsistency easy protect accidental loss

destruction alteration accidental introduc- tion inconsistency easy protect accidental loss datum consistency protect malicious access datum absolute protection information store computer system malicious abuse possible cost perpetrator suffi- ciently high deter attempt access information proper authority type attack launch program individual computer masse stack- buffer overflow tech- nique allow successful attacker change level system access virus malware require human interaction worm self- perpetuate infect thousand computer denial of- service attack prevent legitimate use target system encryption limit domain receiver datum authentication limit domain sender encryption provide confidential- ity datum store transfer symmetric encryption require share key asymmetric encryption provide public key pri- vate key authentication combine hashing prove datum change user authentication method identify legitimate user system addition standard user password protection sev- eral authentication method time password example change session session avoid replay attack factor authen- tication require form authentication hardware calcula- tor activation pin present different response base time multifactor authentication use form method greatly decrease chance authentication forgery method prevent detect security incident include to- date security policy intrusion detection system antivirus software audit- ing logging system event system monitoring code signing sandboxing firewall information virus worm find http://www.securelist com

ludwig 1998 ludwig 2002 website con- taine date security information http://www.eeye.com/resources/se curity center research apaper danger computer monoculture find http://cryptome.org/cyberinsecurity.htm paper discuss privilege multics overview original article explore buffer overflow attack http://phrack.org/issues/49/14.html development version control system git https://github.com/git/. c. kaufman 2002 stallings brown 2011 explore use cryptography computer system discussion concern protection digital signature offer akl 1983 davies 1983 denning 1983 denning 1984 complete cryptography information present schneier 1996 katz lindell 2008 asymmetric key encryption discuss https://www-ee.stanford.edu/ hellman publications/24.pdf tls cryptographic protocol describe detail https://tools.ietf.org/html/rfc5246 nmap network scanning tool http://www.insecure.org/nmap/. information port scan hide http://phrack.org/issues/49/15.html nessus commercial vulnerability scanner free limited target s. g. akl digital signature tutorial survey computer vol- ume 16 number 2 1983 page 15–24 c. kaufman 2002

m. s. c. kaufman r. perlman network security private communication public world second edition prentice hall 2002 d. w. davies apply rsadigital signature electronic mail computer volume 16 number 2 1983 page 55–62 d. e. denning protect public keys signature key computer volume 16 number 2 1983 page 27–35 d. e. denning digital signature rsa pub- lic key cryptosystem communications acm volume 27 number 4 1984 page 388–392 katz lindell 2008 j. katz y. lindell introduction modern cryptog- raphy chapman hall crc press 2008 m. ludwig giant black book computer viruses second edition american eagle publications 1998 m. ludwig little black book email viruses american eagle publications 2002 b. schneier applied cryptography second edition john wiley sons 1996 stallings brown 2011 w. stallings l. brown computer security principles practice second edition prentice hall 2011 chapter 16 exercise 16.1 buffer overflow attack avoid adopt well program- ming methodology special hardware support discuss password know user variety way simple method detect event occur explain answer purpose salt user provide pass- word salt store list password keep operate system user manage read list password protection long provide suggest scheme avoid problem hint use different internal external representation experimental addition unix allow user connect watch- dog program file watchdog invoke program request access file watchdog grant deny access file discuss pro con watchdog discuss means manager system connect inter- net design system limit eliminate damage worm drawback make change list security concern bank computer system item list state concern relate physical human operating system security advantage encrypt datum store computer commonly computer program prone man the-middle attack discuss solution prevent form attack compare symmetric asymmetric encryption scheme discuss circumstance distribute system use dkd n(eke n(m provide authentication sender use encryption discuss asymmetric encryption algorithm achieve following goal authentication receiver know sender generate message

algorithm achieve following goal authentication receiver know sender generate message secrecy receiver decrypt message authentication secrecy receiver decrypt message receiver know sender generate message consider system generate 10 million audit record day assume average 10 attack day system attack reflect 20 record intrusion detection system true alarm rate 0.6 false alarm rate 0.0005 percentage alarm generate system correspond real mobile operate system ios android place user datum system file separate partition aside security advantage separation c h p t e r chapter 16 address security involve guard computer resource unauthorized access malicious destruction alteration accidental introduction inconsistency chapter turn protection involve control access process user resource define computer system process operate system protect activity provide protection use mechanism ensure process gain proper authorization operate system operate file memory segment cpu networking resource system mechanism provide means specify control impose means enforcement discuss goal principle protection modern computer explain protection domain combine access matrix specify resource process access examine capability- language base protection system describe protection mechanism mitigate system attack 17.1 goal protection computer system sophisticated pervasive application need protect integrity grow protection originally conceive adjunct multiprogramme operate system untrustworthy user safely share common logical space directory file common physical space memory modern protection concept evolve increase reliability com- plex system make use share resource connect insecure communication platform internet need provide protection reason obvious need prevent mischievous intentional violation access restriction user general importance need ensure process system use system resource way consistent state policy requirement absolute reliable system protection improve reliability detect latent error interface component subsystem early detection interface error prevent contamination healthy subsystem malfunction subsys- tem unprotected resource defend use misuse unauthorized incompetent user protection orient system provide mean distinguish authorized unauthorized usage role protection computer system provide mechanism enforcement policy govern resource use policy establish variety

provide mechanism enforcement policy govern resource use policy establish variety way fix design system formulate management system define individual user protect resource protection system flexibility enforce variety policy policy resource use vary application change time reason protection long concern solely designer operate system application programmer need use protection mechanism guard resource create support application subsystem misuse chapter describe protection mechanism operate system provide application designer use design protection software note mechanism distinct policy mechanism determine policy decide separation policy mechanism important flexibility policy likely change place place time time bad case change policy require change underlie mechanism general mechanism enable avoid situation 17.2 principle protection frequently guide principle project design operate system follow principle simplify design decision keep system consistent easy understand akey time- test guide principle protection principle privilege discuss chapter 16 principle dictate program user system give privilege perform task consider tenet unix user run root unix root user execute privileged command user innately respect fear accidental delete operation correspond undelete root virtually omnipotent potential human error user act root grave consequence far consider human error damage result malicious attack virus launch accidental click attachment example buffer overflow code injection attack successfully carry root privileged process windows process administrator privilege case prove catastrophic system observe principle privilege system chance mitigate attack malicious code obtain root privilege chance adequately define permission block damage operation sense permission act like immune system operate system level principle privilege take form examine detail later chapter important principle see derivative principle privilege compartmentalization com- partmentalization process protect individual system compo- nent use specific permission access restriction component subvert line defense kick attacker compromise system compartmentalization implement form network demilitarize zone dmzs careful use access restriction help system secure beneficial produce audit trail track divergence allow access audit trail hard record system log monitor closely

allow access audit trail hard record system log monitor closely reveal early warning attack integrity maintain despite attack provide clue attack vector accurately assess damage cause importantly single principle panacea security vulnerability defense depth multiple layer protection apply think castle garrison wall moat protect time course attacker use multiple mean bypass defense depth result escalate arm 17.3 protection rings see main component modern operate system ker- nel manage access system resource hardware kernel definition trusted privileged component run high level privilege user process carry privilege separation hardware support require modern hardware support notion separate execution level implementation vary somewhat popular model privilege separation protection ring model fashion bell define set concentric ring ring provide subset functionality ring j j < i. innermost ring ring 0 provide set privilege pattern show figure 17.1 system boot boot high privilege level code level perform necessary initialization drop privileged level order return high privilege level code usually call special instruction refer gate provide portal ring syscall instruction intel example call instruction shift execution user kernel mode see execute system ring n 1 transfer execution predefine address allow caller specify argument include system number arbitrary kernel address way integrity privileged ring generally assure way end privileged ring occurrence processor trap interrupt occur execution immediately transfer high privilege ring execution high privilege ring predefine restrict guard code intel architecture follow model place user mode code ring 3 kernel mode code ring 0 distinction bit special eflags register access register allow ring 3 prevent- e malicious process escalate privilege advent virtual- ization intel define additional ring -1 allow hypervisor virtual machine manager create run virtual machine hypervisor capability kernel guest operate system arm processor architecture initially allow usr svc mode user kernel supervisor mode respectively armv7 processor arm introduce trustzone tz provide additional ring priv- ileged execution

arm introduce trustzone tz provide additional ring priv- ileged execution environment exclusive access hardware back cryptographic feature nfc secure element chip cryp- tographic key handle password sensitive information secure kernel access chip key request encryption decryption service trustzone environment mean specialized instruction secure monitor smc usable kernel mode system call kernel ability directly execute specific address trustzone pass argu- ment register android use trustzone extensively version 5.0 show figure 17.2 correctly employ trust execution environment mean kernel compromise attacker simply retrieve key kernel memory move cryptographic service separate trust environment domain protection android use trustzone make brute force attack likely succeed describe chapter 16 attack involve try possible combination valid password character password find key system user password system store chip key accessible trust context key password enter verify request trustzone environment key know guess trustzone verifier impose limitation cap number verification attempt example 64 bit armv8 architecture arm extend model support level call exception level number el0 el3 user mode run el0 kernel mode el1 el2 reserve hypervisor el3 privileged reserve secure monitor trustzone layer exception level allow run separate operate system show figure 17.3 note secure monitor run high execution level general- purpose kernel make perfect place deploy code check kernel integrity functionality include samsung realtime kernel protection rkp android apple watchtower know kpp kernel patch protection ios 17.4 domain protection ring protection separate function domain order hierar- chically generalization ring domain hierarchy computer system treat collection process object object mean hardware object cpu memory segment printer disk tape drive software object file program semaphore object unique differentiate object system access define meaningful operation object essentially abstract data type operation possible depend object example cpu execute memory word read write dvd rom read tape drive read write rewind data file create open read write close delete program file

data file create open read write close delete program file read write execute delete process allow access object authorization furthermore time process able access object currently require complete task second requirement need know principle useful limit damage faulty process attacker cause system example process p invoke procedure procedure allow access variable formal parameter pass able access variable process p. similarly consider case process p invoke compiler compile particular file compiler able access file arbitrarily access define subset file source file output object file related file compile conversely compiler private file accounting optimization purpose process p able access compare need know privilege easy think need know policy privilege mechanism achiev- e policy example file permission need know dictate user read access write execute access file principle privilege require operate system provide mechanism allow read write execute access domain protection facilitate sort scheme describe process operate protection domain specify resource process access domain define set object type operation invoke object ability execute operation object access right domain collection access right ordered pair < object right set > example domain d access right < file f read write > process execute domain d read write file f. perform operation object domain share access right example figure 17.4 domain d1 d2 d3 access right < o4 print >

share d2 d3 imply process execute domain print object o4 note process execute domain d1 read write object o1 process domain d3 execute object o1 association process domain static set resource available process fix process lifetime dynamic expect establish dynamic protection domain complicated establish static protection domain association process domain fix want adhere need know principle mechanism available change content domain reason stem fact process execute different phase example need read access phase write access domain static define domain include read write access arrangement provide right need phase read access phase need write access vice versa need know principle violate allow content domain modify domain reflect minimum necessary association dynamic mechanism available allow domain switching enable process switch domain want allow content domain change change content domain provide effect create new domain change content switch new domain want change domain content < o3 read write > < o1 read write > < o2 execute > < o1 execute > < o3 read > < o2 write > < o4 print >

system protection domain domain realize variety way user domain case set object access depend identity user domain switching occur user change generally user log user log process domain case set object access depend identity process domain switching occur process send message process wait procedure domain case set object access correspond local variable define procedure domain switching occur procedure discuss domain switching great detail section 17.5 consider standard dual mode kernel user mode model operating- system execution process kernel mode execute privileged instruction gain complete control computer system con- trast process execute user mode invoke nonprivileged instruction consequently execute predefine memory space mode protect operate system execute kernel domain user process execute user domain multipro- grammed operating system protection domain insufficient user want protect elaborate scheme need illustrate scheme examine influential operate system unix android implement note early unix root user execute privileged command user restrict certain operation root user impair user everyday operation consider example user want change password inevitably require access password database commonly /etc shadow access root similar challenge encounter set schedule job reach normal user solution problem setuid bit unix owner identi- fication domain bit know setuid bit associate file setuid bit enable bit enable executable file chmod + s execute file temporarily assume identity file owner mean user manage create file user id root setuid bit enable gain access execute file user root duration process lifetime strike alarming good reason potential power setuid executable binary expect sterile affect necessary file specific constraint hermetic example tamperproof impossible subvert setuid program need carefully write assurance return example change password passwd command setuid root modify password database present user valid password restrict edit password user user unfortunately experience repeatedly show setuid binary fulfill criterion successfully time setuid binary subvert race condition code injection yield instant root access attacker attacker

race condition code injection yield instant root access attacker attacker frequently successful chapter 16 limit damage bug setuid program discuss example android application id android distinct user id provide application basis application instal installd daemon assign distinct user id uid group id gid private datum directory /data data/<app- >

ownership grant uid gid

combination way application device enjoy level protection provide unix system separate user quick simple way provide isolation security privacy mechanism extend modify kernel allow certain operation networking socket mem- ber particular gid example aid inet 3003 enhancement android define certain uid isolate prevent initiate rpc request bare minimum service 17.5 access matrix general model protection view abstractly matrix call access matrix row access matrix represent domain column represent object entry matrix consist set access right column define object explicitly omit object access right entry access(i j define set operation process execute domain di invoke object oj illustrate concept consider access matrix show figure 17.5 domain object file f1 f2 f3 laser printer process execute domain d1 read file f1 f3 process execute domain d4 privilege execute domain d1 addition write file f1 f3 laser printer access process execute domain d2 access matrix scheme provide mechanism specify variety policy mechanism consist implement access matrix ensure semantic property outline hold specif- ically ensure process execute domain di access object specify row allow access matrix access matrix implement policy decision concern protection policy decision involve right include j)th entry decide domain process execute policy usually decide operate system user normally decide content access matrix entry user create new object oj column oj add access matrix appropriate initialization entry dictate creator user decide enter right entry column j right entry need access matrix provide appropriate mechanism define implement strict control static dynamic association process domain switch process domain execute operation switch object domain control domain switching include domain object access matrix similarly change content access matrix perform operation object access matrix control change include access matrix object actually entry access matrix modify individually consider entry access matrix object protect need consider operation possible new object domain access matrix decide want process able execute process

object domain access matrix decide want process able execute process able switch domain switch domain di domain dj allow access right switch access(i j figure 17.6 process execute domain d2 switch domain d3 domain d4 process domain d4 switch d1 domain d1 switch d2 allow control change content access matrix entry require additional operation copy owner control examine operation ability copy access right domain row access matrix denote asterisk append access right copy right allow access right copy column object right define example figure 17.7(a process execute domain d2 copy read operation entry associate file f2 access matrix figure 17.7(a modify access matrix show figure 17.7(b access matrix figure 17.5 domain object scheme additional variant 1 right copy access(i j access(k j remove access(i j

action transfer right copy 2 propagation copy right limit right ris copy access(i j access(k j right r r create process execute domain dk copy right system select copy right provide identify separate right copy transfer limited access matrix copy right need mechanism allow addition new right removal right owner right control operation access(i j include owner right process execute domain di add remove right entry column j. example figure 17.8(a domain d1 owner f1 add delete valid right column f1 similarly domain d2 owner f2 f3 add remove valid right column access matrix figure 17.8(a modify access matrix show figure 17.8(b copy owner right allow process change entry column mechanism need change entry row control right applicable domain object access(i j include control right process execute domain di remove access right row j.

example suppose figure 17.6 include control right access(d2 d4 process execute domain d2 modify domain d4 show figure 17.9 copy owner right provide mechanism limit prop- agation access right appropriate tool prevent propagation disclosure information problem guarantee information initially hold object migrate outside execution environment call confinemen problem problem general unsolvable bibliographical note end chapter access matrix owner right implementation access matrix modify access matrix figure 17.6 operation domain access matrix them- self important illustrate ability access matrix model let implement control dynamic protection requirement new object new domain create dynamically include access matrix model show basic mechanism exist system designer user policy decision concern domain access object way 17.6 implementation access matrix access matrix implement effectively general matrix sparse entry data- structure technique available represent sparse matrix particularly useful application way protec- tion facility describe method implement access matrix compare method simple implementation access matrix global table consist set ordered triple < domain object right set > operation m execute object oj domain di global table search triple < di oj rk > m rk triple find operation allow continue exception error condition

implementation suffer drawback table usually large keep main memory additional o need virtual memory technique manage table addition difficult advantage special grouping object domain example read particular object object separate entry domain access list object column access matrix implement access list object describe section 13.4.2 obviously entry discard result list object consist order pair < domain right set > define domain nonempty set access right object approach extend easily define list plus default set access right operation m object oj attempt domain di search access list object oj look entry < di rk > m rk entry find allow operation check default set m default set allow access access deny exception condition occur efficiency check

set allow access access deny exception condition occur efficiency check default set search access list capability list domain associate column access matrix object access list associate row domain capability list domain list object operation allow object object represent physical address call capability execute operation m object oj process execute operation m specify capability pointer object oj parameter simple possession capability mean access allow capability list associate domain directly accessible process execute domain capability list protect object maintain operate system access user indirectly capability base protection rely fact capability allow migrate address space directly accessible user process modify capability secure object protect secure unauthorized access capability originally propose kind secure pointer meet need resource protection foresee multiprogramme com- puter system come age idea inherently protect pointer provide foundation protection extend application level provide inherent protection distinguish capability kind object interpret abstract machine high level program run capability usually distinguish datum way object tag denote capability accessible datum tag directly accessible application program hardware firmware support enforce restriction bit necessary distinguish capability object bit extension allow object tag type hardware hardware distinguish integer float point number pointer boolean character instruction capability uninitialized value alternatively address space associate program split part accessible program contain program normal datum instruction contain capability list accessible operate system segmented memory space useful support approach implementation access matrix capability base protection system develop describe briefly section 17.10 mach operate system use version capability base protection describe appendix

d. lock key mechanism lock key scheme compromise access list capability list object list unique bit pattern call lock similarly domain list unique bit pattern call key process execute domain access object domain key match lock object capability list list key domain manage operate system behalf domain user allow examine modify list key lock directly expect choose technique implement access matrix

list key lock directly expect choose technique implement access matrix involve trade off global table simple table large advantage special grouping object domain access list correspond directly need user user create object specify domain access object operation allow access right information particular domain localize determine set access right domain difficult addition access object check require search access list large system long access list search time consume capability list correspond directly need user useful localize information give process process attempt- e access present capability access protection system need verify capability valid revocation capability inefficient section 17.7 lock key mechanism mention compromise access list capability list mechanism effective flexible depend length key key pass freely domain domain addition access privilege effectively revoke simple technique change lock associate object system use combination access list capability process try access object access list search access deny exception condition occur capability create attach process additional reference use capability demonstrate swiftly access allow access capability destroy strategy multics system cal system example strategy work consider file system file associate access list process open file directory structure search find file access permission check buffer allocate information record new entry file table associate process operation return index table newly open file operation file specification index file table entry file table point file buffer file close file table entry delete file table maintain operate system user accidentally corrupt user access file open access check file open protection ensure strategy unix system right access check access file table entry capability allow operation file open reading capability read access place file table entry attempt write file system identify protection violation compare request operation capability file- 17.7 revocation access right dynamic protection system need revoke access right object share different user question revocation immediate versus delay revocation occur immediately delay revocation delay find

immediate versus delay revocation occur immediately delay revocation delay find place selective versus general access right object revoke affect user access right object specify select group user access right revoke partial versus total subset right associate object revoke revoke access right object temporary versus permanent access revoke permanently revoke access right available access revoke later obtain access list scheme revocation easy access list search access right revoke delete list revocation immediate general selective total partial permanent capability present difficult revocation problem mention early capability distribute sys- tem find revoke scheme implement revocation capability include follow reacquisition periodically capability delete domain process want use capability find capability delete process try reacquire capability access revoke process able reacquire capability pointer list pointer maintain object point capability associate object revocation require follow pointer change capability necessary scheme adopt multics system general implementation costly role base access control indirection capability point indirectly directly object capability point unique entry global table turn point object implement revocation search global table desire entry delete access attempt capability find point illegal table entry table entry reuse capability difficulty capability table entry contain unique object object capability table entry match scheme adopt cal system allow selective revocation key key unique bit pattern associate capability key define capability create modify inspect process own capability master key associate object define replace set key operation capability create current value master key associate capability capability exercise key compare master key key match operation allow continue exception condition raise revocation replace master key new value set key operation invalidate previous capability object scheme allow selective revocation master key associate object associate list key object selective revocation implement finally group key global table key capability valid key match key global table implement revocation remove match key table scheme key associate object key associate object provide

key table scheme key associate object key associate object provide maximum flexibility key base scheme operation define key insert list delete list available user particular reasonable allow owner object set key object choice policy decision protection system implement define 17.8 role base access control section 13.4.2 describe access control file file system file directory assign owner group possibly list user entity access control information assign similar function add aspect computer system good example find solaris 10 later version idea advance protection available operate system explicitly add principle privilege role base access control rbac facility revolve privilege privilege right execute system use option system open file write access privilege assign process limit exactly access need perform work privilege program assign role user assign role role base password assign role way user execute role 1 privilege role base access control solaris 10 role enable privilege allow user run program accomplish specific task depict figure 17.10 implementation privilege decrease security risk associate superuser setuid program notice facility similar access matrix describe section 17.5 relationship explore exercise end mandatory access control mac operate system traditionally discretionary access control dac means restrict access file system object dac access control base identity individual user group unix base system dac take form file permission settable chmod chown chgrp windows unix variant allow fine granularity mean access control list acls dac prove insufficient year key weakness lie discretionary nature allow owner resource set modify permission weakness unlimited access allow administrator root user see design leave system vulnerable accidental malicious attack provide defense hacker obtain root privilege need arise strong form protection introduce form mandatory access control mac mac enforce system policy root user modify policy explic- itly allow modification system reboot usually alternate configuration restriction impose mac policy rule pow- erful capability root user resource inaccessible intended owner rule pow- erful capability root user resource inaccessible intended owner modern operate system provide mac dac implementation differ solaris introduce mac trusted solaris 2.5 freebsd dac trustedbsd implementation freebsd 5.0 freebsd implementation adopt apple macos 10.5 serve substrate security feature mac ios implement linux mac implemen- tation selinux project devise nsa integrate distribution microsoft windows join trend windows vista mandatory integrity control heart mac concept label label identifier usually string assign object file device like label apply subject actor process subject request perform operation object request serve operate system perform check define policy dictate give label hold subject allow perform operation label object brief example consider simple set label order accord level privilege unclassified secret secret auser secret clearance able create similarly label process access unclassified secret file secret file user process aware existence secret file operate system filter file operation example display list directory content user process similarly protect way unclassified process able perform ipc request secret secret process way mac label implementation access matrix describe early 17.10 capability base system concept capability base protection introduce early 1970s early research system hydra cap system widely provide interesting proving ground protection theory detail system section a.14.1 section a.14.2 consider contemporary approach capability linux use capability address limitation unix model describe early posix standard group introduce capability posix 1003.1e posix.1e eventually withdraw linux quick adopt capability version 2.2 continue add new development essence linux capability slice power root distinct area represent bit bitmask show figure 17.11

fine- grained control privileged operation achieve toggle bit practice bitmask denote capability permit effective inheritable bitmask apply process thread basis furthermore revoke capability reacquire usual capability posix.1e sequence event process thread start set permit capability voluntarily decrease set execution example open network port thread remove capability port open probably capability direct implementation principle privilege explain early tenet security dictate application user give right require normal operation android base linux utilize capability enable system process notably system server avoid root ownership instead selectively enable operation require linux capability model great improvement traditional unix model inflexible thing bitmap bit represent capability make impossible add capability dynamically require recompile kernel add addition feature apply kernel enforce capability apple system protection take form entitlement entitlement declaratory permission xml property list state permission claim necessary program figure 17.12 process attempt privileged operation figure load kernel extension protection improvement methods < doctype plist public -//apple//dtd plist 1.0//en apple darwin entitlement entitlement check need entitlement present prevent program arbitrarily claim entitlement apple embed entitlement code signature explain section 17.11.4 load process way access code signature process kernel easily query signature particular entitlement verify entitlement simple string matching operation way verifiable authenticate app claim entitlement system entitlement com.apple restrict apple binary 17.11 protection improvement method battle protect system accidental malicious damage esca- late operating system designer implement type protection mechanism level section survey important real world system integrity protection apple introduce macos 10.11 new protection mechanism call system integrity protection sip darwin base operate system use sip restrict access system file resource way root user tamper sip use extend attribute file mark restrict protect system binary debug scrutinize tamper importantly code sign kernel extension permit sip configure allow code sign binary sip root powerful user system far root user manage user file install remove program way replace

root user manage user file install remove program way replace mod- ify operating system component sip implement global inescapable screen process exception allow system bina- rie example fsck kextload show figure 17.12 specifically entitle operation designate purpose recall chapter 2 monolithic system place functionality kernel single file run single address space commonly general purpose operate system kernel monolithic there- fore implicitly trust secure trust boundary rest kernel mode user mode system layer reasonably assume attempt compromise system integrity user mode mean system example attacker try gain access exploit unprotected system imperative implement form system filtering accomplish add code kernel perform inspection system gate restrict caller subset system call deem safe require caller function specific system profile construct individual process linux mechanism seccomp bpf harness berkeley packet filter language load custom pro- file linux proprietary prctl system filtering voluntary effectively enforce call run time library initialize loader transfer control program entry point second form system filtering go deeply inspect argument system form protection consider strong apparently benign system call harbor vulner- ability case linux fast mutex futex system race condition implementation lead attacker control kernel memory overwrite total system compromise mutexe fundamental compo- nent

multitasking system filter challenge encounter approach keep flexible possible time avoid need rebuild kernel change new filter require common occurrence differ need different process flexibility especially important give unpredictable nature vulnerability new vulnerability discover day immediately exploitable attacker approach meet challenge decouple filter implemen- tation

kernel kernel need contain set callout implement specialized driver windows kernel module linux extension darwin external modular com- ponent provide filtering logic update independently kernel component commonly make use specialized profiling lan- guage include build interpreter parser profile decouple code provide human readable editable profile simplify update possible filtering component trust user mode daemon process assist validation logic protection improvement methods sandboxing involve run process environment limit basic system process run credential user start access thing user access run system privilege root process literally system case process need user system privilege example word processor need accept network connection network service provide time day need access file specific set term sandboxing refer practice enforce strict limitation process process set system call privilege allow impose irremovable set restriction process early stage startup execution main function early creation fork system process render unable perform operation outside allowed set way possible prevent process communicate system component result tight compartmentalization mitigate damage system process compromise numerous approach sandboxing java .net example impose sandbox restriction level virtual machine system enforce sandboxing mandatory access control mac policy example android draw selinux policy enhance specific label system property service endpoint sandboxing implement combination multiple mech- anism android find selinux useful lacking effec- tively restrict individual system call late android version nougat o use underlie linux mechanism call seccomp bpf mention early apply system restriction use specialized sys- tem c run time library android bionic call system impose restriction android process party application major vendor apple implement sandboxing appear macos 10.5 tiger seatbelt seatbelt opt mandatory allow require application use apple sandbox base dynamic profile write scheme language provide ability control operation allow block argument capability enable apple create different custom fit profile binary system practice continue day figure 17.13 depict profile example apple sandboxing evolve considerably inception ios

17.13 depict profile example apple sandboxing evolve considerably inception ios variant serve code signing chief protection untrusted party code ios start macos 10.8 macos sandbox mandatory automatically enforce mac store download app recently mention early apple adopt system integrity protection sip macos 10.11 later sip effect system wide platform profile apple enforce start system boot process system process enti- tle perform privileged operation code sign apple trust allow file read metadata literal /var sandbox profile macos daemon deny operation fundamental level system trust program script gen- erally item come operate system trust item change change system update trustworthy executable require special permission user administrator run tool party commercial difficult judge sure tool modify way create currently code signing good tool protection arsenal solve problem code signing digital signing program executa- ble confirm change author create use cryptographic hash section 16.4.1.3 test integrity authen- ticity code signing operating system distribution patch party tool alike operate system include ios windows macos refuse run program fail code signing check enhance system functionality way example apple disable program write obsolete version ios stop signing program download app store 17.12 language base protection degree protection provide computer system usually achieve operate system kernel act security agent inspect validate attempt access protect resource comprehensive access validation source considerable overhead hardware support reduce cost validation allow system designer compromise goal protection satisfy goal difficult flexibility implement protection policy restrict support mechanism provide protection environment large necessary secure great operational operating system complex particularly attempt provide high level user interface goal protection refined designer protection system draw heavily idea originate programming language espe- cially concept abstract datum type object protection system concern identity resource access attempt functional nature access new protection system concern function invoke extend set system define function standard file access method include function user define

define function standard file access method include function user define policy resource use vary depend application subject change time reason protection long consider matter concern designer operate system available tool use application designer resource application subsystem guard tampering influence error point programming language enter picture specify desire control access share resource system make declarative statement resource kind statement integrate language extension typing facility protection declare datum typing designer subsystem specify require- ment protection need use resource system specification give directly program compose language program state approach 1 protection need simply declare program sequence call procedure operate system 2 protection requirement state independently facility pro- vide particular operating system 3 mean enforcement need provide designer 4 declarative notation natural access privilege closely relate linguistic concept datum type variety technique provide programming language implementation enforce protection depend degree support underlie machine operate system example suppose language generate code run cambridge cap system section a.14.2 system storage reference underlie hardware occur indirectly capability restriction prevent process access resource outside protection envi- ronment time program impose arbitrary restriction resource execution particular code segment implement restriction readily software capa- bilitie provide cap language implementation provide standard protect procedure interpret software capability realize protection policy specify language scheme put policy specification disposal programmer free implement enforcement system provide protection kernel powerful hydra section a.14.1 cap mechanism available implement protection specification give programming language principal distinction security protection great support protection kernel mechanism rely assumption operational state system compiler separate reference certify protection violation occur violation possible treat differently security provide form protection rest assumption code generate compiler modify prior execution relative merit enforcement base solely kernel oppose enforcement provide largely compiler security enforcement kernel provide great degree security protection system generation protection- check code compiler compiler support scheme security rest correctness translator underlie mechanism storage management

support scheme security rest correctness translator underlie mechanism storage management protect segment compile code execute ultimately security file program load consideration apply software- support protection kernel less degree kernel reside fix physical storage segment load designate file tag capability system address computation perform hardware fix microprogram great security possible hardware support protection relatively immune protection violation occur result hardware system software malfunction flexibility limit flexibility protection kernel imple- mente user define policy supply adequate facility system provide enforcement policy pro- gramming language protection policy declare enforcement provide need implementation language provide sufficient flexibility extend replace disturbance cause modification operate system kernel efficienc great efficiency obtain enforcement protec- tion support directly hardware microcode insofar software support require language base enforcement advantage static access enforcement verify line compile time intelligent compiler tailor enforcement mechanism meet specify need fix overhead kernel call avoid summary specification protection program language allow high level description policy allocation use resource language implementation provide software protection enforcement automatic hardware support checking unavailable addition interpret protection specification generate call protection system provide hardware operate system way make protection available application program use software capability object com- putation inherent concept idea certain program component privilege create examine software capability capability create program able execute primitive operation seal datum structure render content inaccessible program component hold seal unseal privilege component copy datum structure pass address program component gain access content reason introduce software capability bring protection mechanism programming language problem con- cept propose use seal unseal operation take procedural approach specify protection nonprocedural declarative notation preferable way protection available application need safe dynamic access control mechanism distribut- e capability system resource user process contribute overall reliability system access control mechanism safe use useful practice reasonably efficient require- ment lead development number language construct allow programmer declare restriction use specific man- age

language construct allow programmer declare restriction use specific man- age resource bibliographical note appropriate reference construct provide mechanism function 1 distribute capability safely efficiently customer process particular mechanism ensure user process use manage resource grant capability resource 2 specify type operation particular process invoke allocate resource example reader file allow read file writer able read write necessary grant set right user process impossible process enlarge set access right authorization access control 3 specify order particular process invoke operation resource example file open read possible process different restriction order invoke operation allocate incorporation protection concept programming language practical tool system design infancy protection likely matter great concern designer new system distribute architecture increasingly stringent requirement datum security importance suitable language notation express protection requirement recognize widely run time base enforcement protection java java design run distribute environment java virtual machine jvm build protection mechanism java program compose class collection data field func- tion call method operate field jvm load class response request create instance object class novel useful feature java support dynamically load- e untrusted class network execute mutually distrusting class jvm capability protection paramount concern class run jvm different source equally trust result enforce protection granularity jvm process insufficient intuitively request open file allow generally depend class request open operate system lack knowledge protection decision handle jvm jvm load class assign class protection domain give per- mission class protection domain class assign depend urlfrom class load digital signature class file digital signature cover section 16.4.1.3 config- urable policy file determine permission grant domain class example class load trusted server place protection domain allow access file user home direc- tory class load untrusted server file access permission complicate jvm determine class responsible request access protected resource access perform indirectly system library class example consider class allow open network connection system library request load content url jvm decide open network connection request class determine connection allow application system library philosophy adopt java class determine connection allow application system library philosophy adopt java require library class explicitly permit network connection generally order access protect resource method calling sequence result request take responsibility request presumably perform check necessary ensure safety request course method allow assert privilege method assert privilege class protection domain allow exercise privilege implementation approach call stack inspection thread jvm associate stack ongoing method invocation caller trust method execute access request doprivileged block perform access protect resource directly indirectly doprivileged static method accesscontroller class pass class run method invoke doprivileged block enter stack frame method annotate indicate fact content block execute access protect < request u proxy > resource subsequently request method method call checkpermissions invoke stack inspection determine request allow inspection examine stack frame call thread stack start recently add frame work old stack frame find dopriv- ileged annotation checkpermission return immediately silently allow access stack frame find access disallow base protection domain method class check- permission throw accesscontrolexception stack inspection exhaust stack find type frame access allow depend implementation implementation jvm allow access implementation stack inspection illustrate figure 17.14 gui method class untrusted applet protection domain perform operation open invocation method class url loader protection domain permit open session site lucent.com domain particular proxy server proxy.lucent.com retrieve url reason untrusted applet invocation succeed checkpermission network- e library encounter stack frame method perform open doprivileged block untrusted applet open invocation result exception checkpermissions find doprivileged annotation encounter stack frame course stack inspection work program unable modify annotation stack frame manipulate stack inspection important difference java language include c++ java program directly access memory manipulate object reference reference forge manipulation well- define interface compliance enforce sophisticated collection load time run time check result object manipulate run time stack reference stack compo- nent protection system generally java load time run time check enforce

protection system generally java load time run time check enforce type safety java class type safety ensure class treat integer pointer write past end array access memory arbitrary way program access object method define object class foundation java protection enable class effectively encapsulate protect datum method class load jvm example variable define private class contain access protect access class contain subclass class class package type safety ensure restriction system protection feature guide principle need know implement mechanism enforce principle privilege computer system contain object protect misuse object hardware memory cpu time o device software file program semaphore access right permission perform operation object domain set access right process execute domain use access right domain access manipulate object lifetime process bind protection domain allow switch domain common method secure object provide series protection ring privilege arm example provide protection level privileged trustzone callable kernel mode access matrix general model protection provide mech- anism protection impose particular protection policy system user separation policy mechanism important design property access matrix sparse normally implement access list associate object capability list associate domain include dynamic protection access matrix model consider domain access matrix object revoca- tion access right dynamic protection model typically easy implement access list scheme capability list real system limited general model old unix distribution representative provide discretionary access control read write execution protection separately owner group general public file modern system close general model provide variety protection feature protect system user solaris 10 system implement principle privilege role base access control form access matrix protection extension mandatory access control form system capability base system offer finer grained protection old model provide specific ability process slice power root distinct area method improve protection include system integrity protection system filtering sandboxing code signing language base protection provide fine grained arbitration request privilege operate system

base protection provide fine grained arbitration request privilege operate system able provide example single java jvm run thread different protection class enforce resource request sophisticated stack inspection type safety language concept capability evolve iliffe jodeit codeword implement rice university computer iliffe jodeit 1962 term capability introduce dennis horn 1966 principle separation policy mechanism advocate designer hydra levin et al 1975 use minimal operating system support enforce protection advocate exokernel project ganger et al 2002 kaashoek et al access matrix model protection domain object develop lampson 1969 lampson 1971 popek 1974

saltzer schroeder 1975 provide excellent survey subject posix capability standard way implement linux describe https://www.usenix.org/legacy/event/usenix03/tech/freenix03/ paper gruenbacher gruenbacher html main.html detail posix.1e linux implementation provide dennis horn 1966 j. b. dennis e. c. v. horn programming seman- tics multiprogrammed computations communications acm volume 9 number 3 1966 page 143–155 ganger et al 2002 g. r. ganger d. r. engler m. f. kaashoek h. m. briceno r. hunt t. pinckney fast flexible application level networking exokernel systems acm transactions computer systems volume 20 number 1 2002 page 49–83 iliffe jodeit 1962 j. k. iliffe j. g. jodeit dynamic storage alloca- tion system computer journal volume 5 number 3 1962 page 200–209 kaashoek et al 1997 m. f. kaashoek d. r. engler g. r. ganger h. m. briceno r. hunt d. mazieres t. pinckney r. grimm j. jannotti k. macken- zie application performance flexibility exokernel systems proceed- ing acm symposium operating systems principles 1997 page 52–65 b. w. lampson dynamic protection structures proceedings afips fall joint computer conference 1969 page 27–38 b. w. lampson protection proceedings fifth annual princeton conference information systems science 1971 page 437–443 levin et al 1975 r. levin e. s. cohen w. m. corwin f. j. pollack w. a. wulf policy mechanism separation hydra proceedings acm symposium operating systems principles 1975 page 132–140 g. j. popek protection structure computer volume 7 num- ber 6 1974 page 22–33 saltzer schroeder 1975 j. h. saltzer m. d. schroeder protec- tion information computer systems proceedings ieee 1975 page chapter 17 exercise access control matrix determine process switch domain domain b enjoy access privilege domain b. approach equivalent include access privilege domain b domain consider computer system computer game play student 10 p.m. 6 a.m. faculty member 5 p.m. 8 a.m.

computer center staff time suggest scheme implement policy efficiently hardware feature computer system need efficient capability manipulation feature memory pro- discuss strength weakness implement access matrix access list associate object discuss strength weakness implement access matrix capability associate domain explain capability base system provide great flexibility ring protection scheme enforce protection policy need know principle important protec- tion system adhere principle discuss follow system allow module designer enforce need know principle jvm stack inspection scheme describe java protection model compromise java program allow directly alter annotation stack access matrix facility role base access control facility similar differ principle privilege aid creation protec- system implement principle privilege protection failure lead security violation virtualization permeate aspect computing virtual machine instance trend generally virtual machine guest operat- e system application run environment appear native hardware environment behave native hardware protect manage limit distribute system collection processor share memory clock instead processor local memory processor communicate local area wide area computer network computer network allow disparate compute device communicate adopt standard communica- tion protocol distributed system offer benefit user access resource maintain system boost com- putation speed improve data availability reliability c h p t e r term virtualization meaning aspect virtualization permeate aspect computing virtual machine instance trend generally virtual machine guest operate system applica- tion run environment appear native hardware behave native hardware protect manage limit chapter delve use feature implementation virtual machine virtual machine implement way chapter describe option option add virtual machine support kernel implementation method pertinent book explore fully additionally hardware feature provide cpu o device support virtual machine implementation discuss feature appropriate kernel module explore history benefit virtual machine discuss virtual machine technology describe method implement virtualization identify common hardware feature support virtualization explain operating system module discuss current virtualization research area fundamental idea virtual machine abstract hardware

current virtualization research area fundamental idea virtual machine abstract hardware single computer cpu memory disk drive network interface card forth different execution environment create illusion separate environment run private computer concept similar layered approach operating system implementation section 2.8.2 way case virtualization layer create virtual system operating system application run virtual machine implementation involve component base host underlying hardware system run virtual machine virtual machine manager vmm know hypervisor create run virtual machine provide interface identical host case paravirtualization discuss later guest process provide virtual copy host figure 18.1 usually guest process fact operate system single physical machine run multiple operate system concurrently virtual machine moment note virtualization definition operat- e system blur example consider vmm software vmware esx virtualization software instal hardware run hardware boot provide service application service include traditional one scheduling memory management new type migration application system further- application fact guest operate system vmware esx vmm operate system turn run operate system certainly act like operate system clarity component provide virtual environment vmm implementation vmms vary greatly option include follow hardware base solution provide support virtual machine cre- ation management firmware vmms commonly find mainframe large midsized server generally know system model nonvirtual machine b virtual machine problem computer science solve level indirec- problem layer indirection ”—kevlin operate system like software build provide virtualization include vmware esx mention joyent smartos citrix xenserver vmms know type 1 hypervisor general purpose operating system provide standard function vmm function include microsoft windows server hyperv red hat linux kvm feature system feature set similar type 1 hypervisor know type 1 application run standard operating system provide vmm feature guest operate system application include vmware workstation fusion parallels desktop oracle virtual- box type 2 hypervisor paravirtualization technique guest operate system modify work cooperation vmm optimize performance programming environment virtualization vmms virtu-

work cooperation vmm optimize performance programming environment virtualization vmms virtu- alize real hardware instead create optimize virtual system technique oracle java microsoft net emulator allow application write hardware environment run different hardware environment different type application containment virtualization provide virtualization like feature segregate application contain application make secure manageable variety virtualization technique use today testament breadth depth importance virtualization modern computing virtualization invaluable data center operation efficient application development software testing use evolve available addition original concept find system make worth explore run operating system major difficulty vm approach involve disk system suppose physical machine disk drive want support seven virtual machine clearly allocate disk drive virtual machine solution provide identical system hard disk respect size system implement minidisk allocate track physical disk minidisk need virtual machine create user run oper- ating system software package available underlie interactive operating system remain domain system support virtualization formal definition virtualization help establish system requirement target functionality virtualization requirement fidelity vmm provide environment program essentially identical original machine performance program run environment minor performance decrease safety vmm complete control system resource requirement guide virtualization effort today late 1990s intel 80x86 cpu common fast rich feature accordingly developer launch multiple effort implement virtualization platform xen vmware create technology today allow guest operate system run 80x86 time virtualization expand include common cpu commercial open source tool operating system exam- ple open source virtualbox project http://www.virtualbox.org provide program run intel x86 amd 64 cpu windows linux macos solaris host operating system possible guest operate system include version windows linux solaris bsd include 18.3 benefits features advantage virtualization attractive fundamen- tally relate ability share hardware run different execution environment different operate system concurrently important advantage virtualization host system pro- tecte virtual machine virtual machine protect virus inside guest operate system damage operate system unlikely affect host guest

guest operate system damage operate system unlikely affect host guest benefit feature virtual machine completely isolate virtual machine protection problem potential disadvantage isolation prevent sharing resource approach provide sharing implement possible share file system volume share file second possible define network virtual machine send information virtual communication network network mod- ele physical communication network implement software course vmm free allow number guest use physical resource physical network connection sharing provide vmm case allow guest communicate physical network feature common virtualization implementation ability freeze suspend run virtual machine operate system provide basic feature process vmm step allow copy snapshot guest copy create new vm vm machine current state intact guest resume original machine create clone snapshot record point time guest reset point necessary example change long want vmm allow snapshot take example snapshot record guest state day month make restoration snapshot state possible ability good advantage virtual environment avirtual machine system perfect vehicle operating system research development normally change operate system difficult task operate system large complex program change cause obscure bug appear power operate system make change particularly dangerous operate system execute kernel mode wrong change pointer cause error destroy entire file system necessary test change operate system carefully course operate system run control entire machine system stop take use change test period commonly call system development time make system unavailable user system development time share system schedule late night weekend system load virtual machine system eliminate problem sys- tem programmer give virtual machine system develop- ment virtual machine instead physical machine normal system operation disrupt complete test change ready production advantage virtual machine developer multiple operate system run concurrently developer workstation virtualize workstation allow rapid porting testing program vary environment addition multiple version program run isolate operating system system similarly quality- assurance engineer test application multiple environment with- buy

similarly quality- assurance engineer test application multiple environment with- buy power maintain computer environment major advantage virtual machine production data center use system consolidation involve take separate system run virtual machine system physical virtual conversion result resource optimization lightly system combine create heavily system consider management tool vmm allow system administrator manage system virtual environment include 100 physical server run 20 virtual server virtualization 2,000 server require system administrator virtualization tool work manage administrator tool possible templating standard virtual machine image include instal configure guest operating system application save source multiple run vm feature include manage patching guest back restore guest monitor resource use virtualization improve resource utilization resource management vmm include live migration feature move run- ne guest physical server interrupt opera- tion active network connection server overload live migration free resource source host disrupt guest similarly host hardware repair upgrade guest migrate server evacuate host maintain guest migrate operation occur downtime interruption user think possible effect virtualization application deploy system easily add remove virtual machine install application system directly instead application preinstalle tune customize operate system vir- tual machine method offer benefit application devel- oper application management easy tuning require technical support application straightfor- ward system administrator find environment easy manage installation simple redeploy application system easy usual step uninstalle rein- stall widespread adoption methodology occur format virtual machine standardize virtual machine run virtualization platform open virtual machine format attempt provide standardization succeed unify virtual machine format virtualization lay foundation advance com- puter facility implementation management monitoring cloud comput- ing example possible virtualization resource cpu memory o provide service customer internet technology api program tell cloud compute facility create thousand vm run specific guest operating system application access internet multiuser game photo share site web service use functionality area desktop computing virtualization enable desktop laptop computer user connect

area desktop computing virtualization enable desktop laptop computer user connect remotely virtual machine locate remote datum center access application local practice increase security datum store local disk user site cost user compute resource decrease user networking cpu memory system component need display image guest run remotely protocol rdp need expensive high performance component use virtualization sure follow prevalent hardware support continue improve 18.4 building block virtual machine concept useful difficult implement work require provide exact duplicate underlie machine especially challenge dual mode system underlie machine user mode kernel mode section examine building block need efficient virtualization note building block require type 0 hypervisor discuss section 18.5.2 ability virtualize depend feature provide cpu feature sufficient possible write vmm provide guest environment virtualization impossible vmm use sev- eral technique implement virtualization include trap emulate binary translation discuss technique section hardware support need support virtualization read section mind important concept find virtualization option implementation virtual cpu vcpu vcpu execute code represent state cpu guest machine believe guest vmm maintain vcpu represent guest current cpu state guest context switch cpu vmm information vcpu load right context general purpose operate system use pcb typical dual mode system virtual machine guest execute user mode extra hardware support provide kernel course run kernel mode safe allow user level code run kernel mode physical machine mode virtual machine consequently virtual user mode virtual kernel mode run physical user mode action cause transfer user mode kernel mode real machine system interrupt attempt execute privileged instruction cause transfer virtual user mode virtual kernel mode virtual transfer accomplish procedure follow kernel guest attempt execute privileged instruction error system user mode cause trap vmm real machine vmm gain control execute emulate action attempt guest kernel guest trap emulate virtualization implementation return control virtual machine call trap emulate method show figure 18.2 privileged

virtual machine call trap emulate method show figure 18.2 privileged instruction time issue nonprivileged instruction run natively hardware provide performance guest native application privileged instruction create extra overhead cause guest run slowly natively addition cpu multiprogramme virtual machine slow virtual machine unpredictable way ple allow normal instruction virtual machine execute directly hardware privileged instruction need mainly o emulate execute slowly general evolu- tion hardware performance trap emulate functionality improve case need reduce example cpu extra mode add standard dual mode opera- tion vcpu need track mode guest operating system physical cpu perform function fact cpu provide guest cpu state management hardware vmm need supply functionality remove extra overhead cpu clean separation privileged nonprivileged instruction unfortunately virtualization implementer intel x86 cpu line thought give run virtualization x86 design fact cpu family intel 4004 release 1971 design core calculator chip maintain backward compatibility lifetime prevent change virtualization easy genera- let consider example problem command popf load flag register content stack cpu privileged mode flag replace stack cpu user mode flag replace ignore trap generate popf execute user mode trap emulate procedure render useless x86 instruction cause similar problem purpose discussion set instruction special instruction recently 1998 trap emulate method implement virtualization x86 consider impossible special instruction previously insurmountable problem solve implemen- tation

binary translation technique binary translation fairly simple concept complex implementation basic step follow 1 guest vcpu user mode guest run instruction natively physical cpu 2 guest vcpu kernel mode guest believe run- ne kernel mode vmm examine instruction guest exe- cut virtual kernel mode read instruction guest go execute base guest program counter instruc- tion special instruction run natively special instruction translate new set instruction perform equivalent task example change flag vcpu binary translation show figure 18.3 implement translation code vmm code read native binary instruction dynamically guest demand generate native binary code execute place original code vmm read instruction binary translation virtualization implementation basic method binary translation describe execute correctly perform poorly fortunately vast majority instruction execute natively performance improve instruction turn specific implementation binary translation vmware method way improve performance caching provide solution replacement code instruction need translate cache later execution instruction run translation cache need translate cache large method

greatly improve performance let consider issue virtualization memory management specifically page table vmm page table state guest believe manage page table vmm common method trap emulate binary translation use nest page table npt guest operate system maintain page table translate virtual physical memory vmm maintain npt represent guest page table state create vcpu represent guest cpu state vmm know guest try change page table make equivalent change npt guest cpu vmm put pointer appropriate npt appropriate cpu register table active page table guest need modify page table example fulfil page fault operation intercept vmm appropriate change nested system page table unfortunately use npt cause tlb miss increase complexity need address achieve reasonable performance binary translation method create large amount overhead perform launch new industry aim virtualize intel x86 base system vmware test performance impact binary translation boot system windows xp immediately shut monitor elapsed time number translation produce binary translation method result 950,000 translation take 3 microsecond total increase 3 second 5 percent native execution windows xp achieve result developer performance improvement discuss information consult bibliographical note end chapter level hardware support virtualization impossible hardware support available system feature rich stable virtual machine well perform intel x86 cpu family intel add new virtualization support vt x instruc- tion successive generation begin 2005 binary translation long need fact major general purpose cpu provide extended hardware support virtualization example amd virtualization technology amd- v appear amd processor start 2006 define new mode operation host guest move dual mode multimode processor vmm enable host mode define characteris- tic guest virtual machine switch system guest mode pass control system guest operate system run virtual machine guest mode virtualized operate system think run native hardware see device include host definition guest guest try access virtualize resource control pass vmm manage interaction functionality intel vt x

resource control pass vmm manage interaction functionality intel vt x similar provide root nonroot mode equivalent host guest mode provide guest vcpu state datum structure load save guest cpu state automatically guest context switch addition virtual machine control structure vmcs provide manage guest host state guest execution control exit control information guest exit host case exam- ple nest page table violation cause attempt access unavailable memory result guest exit amd intel address memory management virtual environment amd rvi intel ept memory management enhance- ment vmm long need implement software npt essence cpu implement nest page table hardware allow vmm fully control paging cpu accelerate translation virtual phys- ical address npt add new layer represent guest view logical physical address translation cpu page table walking func- tion traverse datum structure find desire datum include new layer necessary walk guest table vmm table find physical address desire tlb miss result performance penalty table guest host page table traverse complete lookup figure 18.4 show extra translation work perform hardware translate guest virtual address final physical address o area improve hardware assistance consider standard direct memory access dma controller accept target memory address source o device transfer datum operate system action hardware assistance guest try set dma transfer affect memory vmm guest cpu provide hardware assist dma intel cpu vt d dma level indirection vmm set protection domain tell cpu

physical memory belong guest assign o device protection domain allow direct access memory region region hardware transform address dma request issue o device host physical memory address associate o. manner dma transfer pass guest device vmm interference similarly interrupt deliver appropriate guest visible guest provide interrupt remapping feature cpu virtualization hardware assistance automatically deliver

inter- rupt destine guest core currently run thread guest way guest receive interrupt need vmm intercede delivery interrupt remapping malicious guest generate interrupt gain control host system bibliographical note end chapter detail vmm nested page table datum structure host physical address guest virtual address kernel paging datum guest physical address nest page table arm architecture specifically arm v8 64 bit slightly different approach hardware support virtualization provide entire excep- tion level el2 privileged kernel el1 allow running secluded hypervisor mmu access interrupt trapping allow paravirtualization special instruction hvc add allow hypervisor call guest kernel instruction call kernel mode el1 interesting effect hardware assist virtualization allow creation thin hypervisor good example macos hyper- visor framework hypervisor.framework operating system- supply library allow creation virtual machine line types vms implementation code actual work system call kernel privileged virtualization cpu instruction behalf hypervisor process allow management virtual machine hypervisor need load kernel module execute call 18.5 type vm implementation look technique implement virtualization consider major type virtual machine implementation functionality use building block describe create virtual environment course hardware virtual machine run cause great variation implementation method discuss implementation general understanding vmms advantage hardware assistance available virtual machine life cycle let begin virtual machine life cycle hypervisor type time virtual machine create creator give vmm certain parameter parameter usually include number cpu memory networking detail storage detail vmm account create guest example user want create new guest virtual cpu 4 gb memory 10 gb disk space network interface get ip address dhcp access dvd drive vmm create virtual machine parameter case type 0 hypervisor resource usually dedicate situa- tion

virtual cpu available unallocated creation request example fail hypervisor type resource dedicate virtualize depend type certainly ip address can- share virtual cpu usually multiplexe physical cpu discuss section 18.6.1 similarly memory management usually involve allocate memory guest actually exist physical memory complicated describe section 18.6.2 finally virtual machine long need delete happen vmm free disk space remove configuration associate virtual machine essentially forget virtual machine step simple compare build configure run- ning remove physical machine create virtual machine exist easy click clone button provide new ip address ease creation lead virtual machine sprawl occur virtual machine system use history state confusing difficult track type 0 hypervisor type 0 hypervisor exist year name include partition domain hardware feature bring positive negative operate system need special advantage feature vmm encode firmware hypervisor firmware type 0 hypervisor load boot time turn load guest image run partition feature set type 0 hypervisor tend small type implement hardware example system split virtual system dedicate cpu memory o device guest believe dedicated hardware simplify implementation detail o present difficulty easy dedicate o device guest system ethernet port guest example guest o device system provide o device sharing case hypervisor manage share access grant device control partition control partition guest operate system provide service net- working daemon guest hypervisor route o request appropriately type 0 hypervisor sophisticated physical cpu memory run guest case guest paravirtualized aware virtualization assist exe- cution example guest watch signal hardware vmm hardware change occur probe hardware device detect change add subtract cpu memory available resource type 0 virtualization close raw hardware execution consider separately method discuss atype 0 hypervisor run multiple guest operate system hardware partition guest run raw hardware turn vmms essentially guest operate system type 0 hypervisor native operating system subset hardware available guest operate system figure 18.5 type hypervisor

subset hardware available guest operate system figure 18.5 type hypervisor usually provide virtualization within- type 1 hypervisor type 1 hypervisor commonly find company datum center sense data center operating system special purpose operating system run natively hardware provide types vms implementations system call interface run program create run manage guest operate system addition run standard hard- ware run type 0 hypervisor type 1 hypervisor platform guest generally know run native hardware type 1 hypervisor run kernel mode take advantage hardware pro- tection host cpu allow use multiple mode guest oper- ating system control improved performance implement device driver hardware run component operating system provide cpu schedul- e memory management o management protection security frequently provide api api support application guest external application supply feature like backup monitoring secu- rity type 1 hypervisor closed source commercial offering vmware esx open source hybrid open closed source citrix xenserver open xen counterpart type 1 hypervisor data center manager control man- age operate system application new sophisticated way important benefit ability consolidate operate system application few system example have system run 10 percent utilization data center server man- age entire load utilization increase guest application move load system live interruption service snap- shot cloning system save state guest duplicate state easy task restore backup instal manually script tool price increase manageability cost vmm commercial product need learn new management tool method increase complexity type type 1 hypervisor include general purpose oper- ating system vmm functionality operate system red-hat enterprise linux windows oracle solaris perform normal duty provide vmm allow operate system run guest extra duty hypervisor typically provide few virtual- ization feature type 1 hypervisor way treat guest operating system process provide special handling guest try execute special instruction type 2 hypervisor type 2 hypervisor interesting operating system explorer little operating system involvement application- level virtual machine manager type vmm simply process run manage host host know virtual- ization happen vmm type 2 hypervisor limit associate type example user need administrative privilege access hardware assistance feature modern cpu vmm run standard user additional privilege vmm advantage feature limitation extra overhead run update guest os guest os request queue descriptor queue vm accept xen outstanding descriptor descriptor slot await response xen response queue descriptor return xen response service request xen o share circular buffer.1

general purpose operating system guest operating system type 2 hypervisor tend poor overall performance type 0 type 1 case limitation type 2 hypervisor provide benefit run variety general purpose operating system run require change host operating system student use type 2 hypervisor example test non native operating system replace native operating system fact apple laptop student version windows linux unix common operating system available learning experimentation see paravirtualization work differently type virtualization try trick guest operate system believe system paravirtualization present guest system similar identical guest prefer system guest modify run paravirtualized virtual hardware gain extra work efficient use resource small virtualization layer xen vmm leader paravirtulization implement technique optimize performance guest host system example mention early vmms present virtual device guest appear real device instead take approach xen vmm present clean simple device abstraction allow efficient o good o relate communication guest vmm 1barham paul xen art virtualization sosp 03 proceedings nineteenth acm symposium operating systems principles p 164 177 c2003 association computing machinery types vms implementation device guest circular buffer share guest vmm share memory read write datum place buffer show figure 18.6 memory management xen implement nested page table guest set page table set read xen require guest use specific mechanism hypercall guest hyper- visor vmm page table change need mean guest operate system kernel code change default code xen specific method optimize performance xen allow guest queue multiple page table change asynchronously hypercall check ensure change complete continue xen allow virtualization x86 cpu use binary trans- lation instead require modification guest operate system like describe time xen take advantage hardware feature support virtualization result long require modify guest essentially need paravirtualization method paravirtualization solution type 0 hypervisor kind virtualization base different execution model virtualization programming environment programming language design run custom build virtualized environment exam- ple

programming language design run custom build virtualized environment exam- ple oracle java feature depend run java virtual machine jvm include specific method security memory define virtualization include duplication hardware virtualization need limit definition instead define virtual environment base api provide set feature want available particular language program write language java program run jvm environment jvm compile native program system run arrangement mean java program write run system include major operate system jvm available say interpret language run inside program read instruction interpret native operation virtualization probably common method run application design operate system different operate system cpu method work relatively efficiently application compile instruction set target system use application operating system need run different cpu necessary translate source cpu instruction turn equivalent instruction target cpu environment long virtualize fully emulate emulation useful host system system architecture guest system compile different architecture example suppose company replace outdated computer system new system like continue run certain important program compile old system program run emulator translate outdated system instruction native instruction set new system emulation increase life program allow explore old architecture have actual old machine expect major challenge emulation performance instruction set emulation run order magnitude slow native instruction instruction new system read parse simulate instruction old system new machine time fast old program run new machine run slowly native hardware challenge emulator writer difficult create correct emulator essence task involve write entire cpu software spite challenge emulation popular particularly gaming circle popular video game write platform long production user want run game frequently find emulator platform run game unmodified emulator modern system fast old game console apple iphone game emulator game available run goal virtualization instance provide method segregate application manage performance resource use create easy way start stop manage case fledge virtualization need application compile operate system need complete virtualization

fledge virtualization need application compile operate system need complete virtualization provide feature instead use application containment consider example application containment start version 10 oracle solaris include container zone create virtual layer operate system application system kernel instal hardware virtualize operate system device virtualize provide process zone impression process system container create application network stack network address port user account cpu memory resource divide zone system wide process zone fact run scheduler optimize performance application allot resource figure 18.7 show solaris 10 system container standard global user space container light weight virtualization method use few system resource fast instantiate destroy similar process virtual machine reason commonly especially cloud computing freebsd operate system include container like feature call jails aix similar feature linux add lxc container feature 2014 include common linux distribution virtualization operating system components solaris 10 zone flag clone system source code lxcis available containers easy automate manage lead orchestration tool like docker kubernetes orchestration tool mean automat- e coordinate system service aim simple run entire suite distribute application operate system simple run single program tool offer rapid deployment application consist process container offer monitoring administration feature docker https://www.docker.com/what-docker information kubernetes find https://kubernetes.io/docs/concepts/overview/what-is-kubernetes 18.6 virtualization operating system component far explore building block virtualization var- ious type virtualization section deep dive operating system aspect virtualization include vmm provide core operating system function like scheduling o memory manage- ment answer question vmm schedule cpu use guest operate system believe dedicated cpu memory management work guest require large amount system virtualization single cpu system frequently act like multiprocessor system virtualization software present virtual cpu virtual machine run system schedule use physical cpu virtual machine significant variation virtualization technology diffi- cult summarize effect virtualization scheduling let consider general case vmm scheduling vmm number physical cpu available number thread run cpu thread vmm thread guest

cpu available number thread run cpu thread vmm thread guest thread guest configure certain number virtual cpu creation time number adjust life vm cpu allocate request number guest vmm treat cpu dedicate schedule give guest thread guest cpu situation guest act like native operate system run native cpu course situation cpu vmm need cpu cycle guest management o man- agement steal cycle guest schedule thread system cpu impact action relatively minor difficult case overcommitment guest configure cpu exist system vmm use standard scheduling algorithm progress thread add fairness aspect algorithm example hardware cpu guest-allocate cpu vmm allocate cpu resource proportionally give guest half cpu resource believe vmm present virtual cpu guest map physical cpu vmm use scheduler distribute appropriately give scheduler provide fairness guest operating system scheduling algorithm assume certain progress give time likely negatively affect virtualization con- sider time share operating system try allot 100 millisecond time slice user reasonable response time virtual machine operate system receive cpu resource virtualization sys- tem give 100 millisecond time slice 100 millisecond virtual cpu time depend busy system time slice second result poor response time user log virtual machine effect real time operating system net outcome scheduling individual virtualize operate system receive portion available cpu cycle believe receive cycle schedule cycle commonly time day clock virtual machine incorrect timer long trigger dedicated cpu virtualization undo scheduling algorithm effort operate system virtual machine correct vmm make application available type operating system system administrator install guest application correct clock drift function virtual virtualization operating system components efficient memory use general purpose operate system major key performance virtualize environment user memory guest application vmm lead pressure memory use add pressure fact vmms typically overcommit memory total memory allocate guest exceed physically exist system extra need efficient memory use lose implementer vmms extensive measure

extra need efficient memory use lose implementer vmms extensive measure ensure optimal use memory example vmware esx use method memory management memory optimization occur vmm establish real memory guest use vmm evaluate guest maximum memory size general purpose operate system expect memory system change vmms maintain illusion guest memory vmm compute target real memory allocation guest base configure memory guest factor overcommitment system load use low level mechanism list reclaim memory 1 recall guest believe control memory allocation page- table management reality vmm maintain nest page table translate

guest page table real page table vmm use extra level indirection optimize guest use mem- ory guest knowledge help approach provide double paging vmm page replacement algorithm load page backing store guest believe physical memory course vmm know guest memory access pattern guest paging efficient create per- formance problem

vmm use method method available provide free memory preferred approach 2 common solution vmm install guest pseudo device driver kernel module vmm control apseudo device driver use device driver interface appear kernel device driver actually control device easy way add kernel mode code directly modify kernel balloon memory manager communicate vmm tell allocate deallocate memory tell allocate allocate memory tell operate system pin allocate page physical memory recall pin lock page physical memory move page guest pin page appear decrease physical memory available create memory pressure guest free physical memory sure free memory vmm know page pin balloon process remove physical page guest allocate guest time guest memory management paging algorithm manage available memory efficient option memory pressure entire system decrease vmm tell balloon process guest unpin free memory allow guest page use 3 common method reduce memory pressure vmm determine page load case vmm reduce number copy page map user page copy vmware example randomly sample guest memory create hash page sample hash value thumbprint page hash page examine compare hash store hash table match page compare byte byte identical page free logical address map physical address technique ineffective consider guest run operating system multiple guest run operate system copy active operating system page need memory similarly multiple guest run set application likely source memory sharing overall effect mechanism enable guest behave perform memory request reality area o hypervisor leeway concern represent underlie hardware guest wide variation o device operate system deal varying flexible o mechanism example operate system device driver mechanism provide uniform interface operate sys- tem o device device driver interface design allow party hardware manufacturer provide device driver connect device operate system usually device driver dynamically load unload virtualization take advantage build flexibility provide specific virtualize device guest operate system describe section 18.5 vmms vary greatly

device guest operate system describe section 18.5 vmms vary greatly provide o guest o device dedicate guest example vmm device driver map guest o. vmm provide idealized device driver guest case guest see easy control device reality simple device driver communicate vmm send request complicated real device complex real device driver o virtual environment complicated require careful vmm design implementation consider case hypervisor hardware combination allow device dedicate guest allow guest access device directly course device dedicate guest available guest direct access useful circumstance reason allow direct access improve o performance hypervisor enable o guest fast o occur type 0 hypervisor provide direct device access guest virtualization operating system components run speed native operate system type 0 hypervisor instead provide share device performance suffer direct device access type 1 2 hypervisor performance similar native operating system certain hardware support present hardware need provide dma pass facility like vt d direct interrupt delivery interrupt go directly guest give frequently interrupt occur surprise guest hardware feature bad performance run natively addition direct access vmms provide share access device con- sider disk drive multiple guest access vmm provide protection device share assure guest access block specify guest configuration instance vmm o check correctness route datum appropriate device guest area networking vmms work general purpose operate system typically internet protocol ip address example connect manage- ment network backup network production network virtualization guest need ip address guest main mode communication server run vmm dozen address vmm act virtual switch route network packet address guest guest directly connect network ip address see broad network know bridging alternatively vmm provide network address translation nat address nat address local server guest run vmm provide routing broad network guest vmm provide firewalling guard connection guest system guest external system important question determine virtualization work multiple operate system instal

system important question determine virtualization work multiple operate system instal boot disk clearly virtualized environment need approach storage management differently native operate system standard multiboot method slice boot disk partition instal boot manager partition instal operate system partition sufficient partitioning limit prevent work ten hundred virtual machine solution problem depend type hypervisor type 0 hypervisor allow root disk partitioning partly system tend run few guest system alternatively disk manager control partition disk manager provide disk space include boot disk partition type 1 hypervisor store guest root disk configuration informa- tion file file system provide vmm type 2 hypervisor store information host operate system file sys- tem essence disk image contain content root disk guest contain file vmm aside potential per- formance problem cause clever solution simplify copy move guest administrator want duplicate guest testing example simply copy associate disk image guest tell vmm new copy boot new virtual machine bring identical guest move virtual machine system run vmm simple halt guest copy image system start guest guest need disk space available root disk image example nonvirtualized database server use file system spread disk store part database virtualize database usually involve create file have vmm present guest disk guest execute usual vmm translate disk o request come guest file o command correct file frequently vmm provide mechanism capture physical system currently configure convert guest vmm manage run physical virtual p v conversion read disk block physical system disk store file vmm system share storage vmm access vmm provide virtual to- physical v p procedure convert guest physical system procedure need debugging problem cause vmm associated component administrator attempt solve problem remove virtualization problem variable v to- p conversion file contain guest datum generate disk block physical disk recreate guest native operate system application testing conclude original system reuse purpose virtual machine

system application testing conclude original system reuse purpose virtual machine return service virtual machine delete original system continue run feature find general purpose operate system find type 0 type 1 hypervisor live migration run guest system mention capability early explore detail live migration work vmm implement relatively easily general purpose operate system spite research attempt let consider live migration work run guest system copy system run vmm copy occur little interruption service user log guest network connection guest continue noticeable impact astonishing ability powerful resource management hard- ware administration compare step necessary virtualization warn user shut process possibly binary restart process new system user access service live migration decrease load overloaded system hardware system change discernable disruption user virtualization operating system components live migration possible define interface guest vmm limited state vmm maintain guest vmm migrate guest following step 1 source vmm establish connection target vmm con- firm allow send guest 2 target create new guest create new vcpu new nest page table state storage 3 source send read memory page target 4 source send read write page target mark 5 source repeat step 4 step page probably modify guest dirty page need send mark clean 6 cycle step 4 5 short source vmm freeze guest send vcpu final state state detail final dirty page tell target start run guest target acknowledge guest run source terminate sequence show figure 18.8

conclude discussion interesting detail limitation concern live migration network connection continue uninter- rupte network infrastructure need understand mac address hardware networking address system virtu- alization happen mac address tie physical hard- ware virtualization mac movable existing networking connection continue reset modern network switch under- stand route traffic mac address accommodate guest target run 5 send dirty page repeatedly 4 send r w page 3 send r o page 1 establish 0 run 7 terminate 2 create 6 run live migration guest server alimitation live migration disk state transfer reason live migration possible guest state maintain guest example open file table system state kernel state disk o slow memory access disk space usually large memory disk associate guest move live migration disk remote guest access network case disk access state maintain guest network connection matter vmm network connection maintain migration remote disk access continue typically nfs cifs iscsi store virtual machine image storage guest need access network base storage access simply continue network connection continue guest migrate live migration make possible manage datum center entirely new way example virtualization management tool monitor vmms environment automatically balance resource use move guest vmm tool optimize use electricity cooling migrate guest select server server handle load power select server entirely load increase tool power server migrate guest despite advantage virtual machine receive little attention number year develop today virtual machine come great use means solve system compat- ibility problem section explore popular contemporary

virtual machine vmware workstation java virtual machine virtual machine typically run operate system design type discuss early chapter vmware workstation popular commercial application abstract intel x86 compatible hardware isolated virtual machine vmware work- station prime example type 2 hypervisor run application host operate system windows linux allow host system run different guest operate system concurrently independent architecture system show figure 18.9 scenario linux run host operating system freebsd windows nt windows xp run guest operate system heart vmware virtualization layer abstract physical hardware isolate virtual machine run guest operate system virtual machine virtual cpu memory disk drive network interface forth physical disk guest own manage file file system host operate system create identical guest simply copy file copy file location protect guest disaster original site move file location host operating system vmware workstation architecture move guest system capability explain early improve efficiency system administration system resource use java virtual machine java popular object orient programming language introduce sun microsystems 1995 addition language specification large api library java provide specification java virtual machine jvm java example programming environment virtualization discuss section 18.5.6 java object specify class construct java program con- sist class java class compiler produce architecture neutral bytecode output .class file run imple- mentation jvm jvm specification abstract computer consist class loader java interpreter execute architecture neutral bytecode diagram figure 18.10 class loader load compile .class file java program java api execution java interpreter class load verifier check .class file valid java bytecode overflow underflow stack ensure bytecode perform pointer arithmetic provide illegal memory access class pass verification run java interpreter jvm automatically manage memory perform garbage collection practice reclaim memory object long use return system research focus garbage collec- tion algorithm increase performance java program virtual windows linux etc java virtual machine jvm implement software host operating system windows

java virtual machine jvm implement software host operating system windows linux macos web browser alternatively jvm implement hardware chip specifically design run java program jvm implement software java interpreter interpret bytecode operation time fast software technique use time jit compiler time java method invoke bytecode method turn native machine language host system operation cache subsequent invocation method perform native machine instruction bytecode operation need interpret run jvm hardware potentially fast special java chip execute java bytecode operation native code bypass need software interpreter time compiler 18.8 virtualization research mention early machine virtualization enjoy grow popularity recent year means solve system compatibility problem research expand cover use machine virtualization include support microservice run library operate system secure partitioning resource embed system consequently lot interesting active research underway frequently context cloud computing application run thousand system well manage deployment virtualize consider execution stack case application service rich general purpose operate system virtual machine manage hypervisor project like unikernels build library operating system aim improve efficiency security environment unikernel specialized machine image address space shrink attack surface resource footprint deploy application essence compile application system library call kernel service use single binary run virtual environment bare metal research change operate system kernel hardware application interact new example cloud computing virtualization create renew interest area http://unikernel.org detail virtualization instruction modern cpu give rise new branch virtualization research focus efficient use hardware well control process partition hypervisor partition exist machine physical resource guest fully commit overcommitte machine resource partition hypervisor securely extend feature exist operate system functionality operate system run separate guest vm domain run subset machine physical resource avoid tedium write entire operate system scratch example linux system lack real time capability safety- security critical task extend lightweight real time operate system run virtual machine traditional hypervisor high overhead run native task new type

machine traditional hypervisor high overhead run native task new type hypervisor need task run virtual machine hypervisor initialize system start task involve continue operation virtual machine allocate hardware free manage hardware interference hypervisor hypervisor interrupt task operation call task task real time aspect secure class partition hypervisor quest v evm xtratum siemens jailhouse project separation hypervisor virtualization partition

separate system component chip level distribute system secure share memory channel implement hardware extend page table separate sandboxe guest communicate target project area robotic self drive car internet things https://www.cs.bu.edu/richwest/papers/west-tocs16.pdf detail virtualization method provide guest duplicate system underlie hardware multiple guest run give system believe native operate system control thank improvement system cpu performance innovative software technique virtualization common fea- ture datum center personal computer pop- ularity cpu designer add feature support virtualization snowball effect likely continue virtualization hardware support increase time virtual machine manager hypervisor create run virtual machine type 0 hypervisor implement hardware require modification operate system ensure proper operation type 0 hypervisor offer example paravirtualization operate system aware virtualization assist execution type 1 hypervisor provide environment feature need cre- ate run manage guest virtual machine guest include software typically associate native system include operate system device driver application user account type 2 hypervisor simply application run operate system know virtualization take place hypervisor hardware host support perform virtualization activity context process programming environment virtualization design programming language language specify contain application program run application provide service pro- emulation host system architecture guest compile different architecture instruction guest want execute translate instruction set native hardware method involve performance penalty balance usefulness able run old program new incompatible hardware run game design old console modern hardware implement virtualization challenging especially hardware support minimal feature provide system easy virtualization implement well performance guest vmm advantage hardware support available optimize cpu scheduling memory management o module provide guest optimum resource use protect vmm guest guest current research extend use virtualization unikernel aim increase efficiency decrease security attack surface compile application library kernel resource application need binary address space run virtual machine partition hypervisor provide secure execution real time operation feature traditionally available application run popek goldberg 1974 establish characteristic help

traditionally available application run popek goldberg 1974 establish characteristic help define vmm method implement virtual machine discuss

agesen et al 2010 intel x86 hardware virtualization support describe neiger et al 2006 amd hardware virtualization support describe white paper available http://developer.amd.com/assets/npt-wp-1%201-final-tm.pdf memory management vmware describe waldspurger 2002 gordon et al 2012 propose solution problem o overhead virtualize environment protection challenge attack virtual environment discuss wojtczuk ruthkowska 2011 early work alternative kernel design https://pdos.csail.mit.edu /6.828/2005 reading engler95exokernel.pdf unikernel west et al 2016 http://unikernel.org partition hypervisor dis- cuss http://ethdocs.org/en/latest/introduction/what-is-ethereum.html https://lwn.net/articles/578295 madhavapeddy et al 2013 quest- v separation hypervisor detail http://www.csl.sri.com/users/rushby/ papers sosp81.pdf https://www.cs.bu.edu/ richwest papers west tocs16 open source virtualbox project available http://www.virtualbox .org source code lxc available https://linuxcontainers.org/lxc/dow docker https://www.docker.com/what-docker informa- tion kubernetes find https://kubernetes.io/docs/concepts/ov agesen et al 2010 o. agesen a. garthwaite j. sheldon p. subrah- manyam evolution x86 virtual machine monitor proceedings acm symposium operating systems principles 2010 page 3–18 gordon et al 2012 a. gordon n. a. n. har'el m. ben yehuda a. landau a. schuster d. tsafrir eli bare metal performance o virtualization proceedings international conference architectural support programming languages operating systems 2012 page 411–422 madhavapeddy et al 2013 a. madhavapeddy r. mirtier c. rotsos d. scott b. singh t. gazagnaire s. smith s. hand j. crowcroft unikernel library operating systems cloud 2013 meyer seawright 1970 r. a. meyer l. h. seawright virtual 1970 page 199–218 neiger et al 2006 g. neiger a. santoni f. leung d. rodgers r. uhlig intel virtualization technology hardware support efficient processor vir- tualization intel technology journal volume 10 2006 popek goldberg 1974 g. j. popek r. p. goldberg formal require- ment virtualizable generation architecture communications acm volume 17 number 7 1974 page 412–421 c. waldspurger memory resource management vmware esx server operating systems review volume 36 number 4 2002 west et al 2016 r. west y. li e. missimer m. danish

virtualized separation kernel mixed criticality systems volume 34 2016 wojtczuk ruthkowska 2011 r. wojtczuk j. ruthkowska follow- e white rabbit software attacks intel vt d technology invisible things lab blog 2011 chapter 18

exercise describe type traditional hypervisor describe virtualization like execution environment explain differ true virtualization describe benefit virtualization vmm unable implement trap emulate base virtual- ization cpu lack ability trap emulate method vmm use implement virtualization hardware assistance virtualization provide mod- live migration possible virtual environment possible native operate system c h p t e r update sarah diesburg distribute system collection processor share memory clock instead node local memory node communi- cate network high speed bus distributed system relevant cer- tainly sort distribute service application distribute sys- tem range provide transparent access file inside organization large scale cloud file photo storage service business analysis trend large datum set parallel processing scientific datum fact basic example distribute system likely familiar chapter discuss general structure distribute system network interconnect contrast main differ- ence type role current distribute system design finally investigate basic design design challenge distribute file explain advantage networked distribute system provide high level overview network interconnect distribute define role type distribute system use today discuss issue concern design distribute file system 19.1 advantage distributed systems distribute system collection loosely couple node interconnect communication network point view specific node distribute system rest node respective resource remote resource local networks distributed systems client server distribute system node distribute system vary size function include small microprocessor personal computer large general- purpose computer system processor refer number name processor site machine host depend context mention mainly use site indicate location machine node refer specific system site node exist client server configuration peer peer configuration hybrid common client server configuration node site server resource node client user like use general structure client server distribute system show figure 19.1 peer peer configuration server client instead node share equal responsibility act client server site connect communication net- work user site opportunity exchange information low level message pass

work user site opportunity exchange information low level message pass system message pass process single computer message system

discuss section 3.4 give message passing high level functionality find standalone system expand encompass distribute system function include file storage execution application remote pro- cedure call rpc major reason build distribute system resource sharing computational speedup reliability section briefly discuss number different site different capability connect user site able use resource available example user site query database locate site b. user site b access file reside site a.

general resource sharing distribute system provide mechanism share file remote site process information distribute database print file remote site remote specialized hardware device supercomputer graphic processing unit gpu perform particular computation partition subcomputation run concurrently distribute system allow distribute sub- computation site subcomputation run con- currently provide computation speedup especially relevant amount customer datum trend addition particular site cur- rently overload request move reroute lightly load site movement job call load balancing common distribute system node service provide internet site fail distribute system remain site continue oper- ating give system well reliability system compose multi- ple large autonomous installation general purpose computer failure affect rest system compose diversified machine responsible cru- cial system function web server file system single failure halt operation system general redundancy hardware datum system continue operation node fail failure node site detect system appro- priate action need recover failure system long use service site addition function fail site take site system ensure transfer function occur correctly finally fail site recover repair mechanism available integrate system smoothly 19.2 network structure completely understand role type distribute system use today need understand network interconnect section serve network primer introduce basic networking concept chal- lenge relate distribute system rest chapter specifically discuss distribute system basically type network local area network lan wide area network wan main difference way geographically distribute local area network compose host distribute small area single building number adjacent building wide area network compose system distribute large area united states difference networks distributed systems imply major variation speed reliability communication network reflect distribute system design local area network emerge early 1970s substitute large mainframe computer system enterprise economi- cal number small computer self contain application single large system small com- puter likely need complement peripheral device disk printer form data sharing likely occur single enterprise natural step

printer form data sharing likely occur single enterprise natural step connect small system lan mention usually design cover small geographical area generally office home environment site system close communication link tend high speed low error rate counterpart wide area typical lan consist number different computer includ- e workstation server laptop tablet smartphone share peripheral device printer storage array router specialized network communication processor provide access network figure 19.2 ethernet wifi commonly con- struct lan wireless access point connect device lan wirelessly router ethernet network generally find business organization computer peripheral tend nonmobile network use coaxial twist pair and/or fiber optic cable send signal ethernet network central controller multiaccess bus new host add easily network ethernet protocol define ieee 802.3 standard typical ethernet speed common twist pair cabling vary 10 mbp 10 gbp type cabling reach speed 100 gbp wifi ubiquitous supplement traditional ethernet net- work exist specifically wifi allow construct network physical cable host wireless transmitter receiver use participate network wifi define ieee 802.11 standard wireless network popular home business public area library internet cafe sport arena bus airplane wifi speed vary 11 mbp 400 mbp ieee 802.3 802.11 standard constantly evolve late information standard speed reference end chapter wide area network emerge late 1960 mainly academic research project provide efficient communication site allow hardware software share conveniently economically wide commu- nity user wan design develop arpanet begin 1968 arpanet grow site experimental network worldwide network network internet know world wide web comprise million computer system site wan physically distribute large geographical area typical link telephone line lease dedicate data line optical cable microwave link radio wave satellite channel communication link control router figure

19.3 responsible direct traffic router network transfer information example internet wan enable host geographically separate site communicate host computer typically differ speed cpu type operating system

communicate host computer typically differ speed cpu type operating system host host operating system communication processor wide area network network distributed systems generally lan turn connect internet regional network regional network interlink router form worldwide network residence connect internet tele- phone cable specialized internet service provider install router connect residence central service course wans internet acompany example

create private wan increase security performance reliability wan generally slow lan backbone wan connec- tion link major city fast transfer rate fiber optic cable fact backbone provider fiber optic speed 40 gbp 100 gbp generally link local internet service providers isps home business slow thing wan link constantly update fast technology demand speed continue grow frequently wans lan interconnect difficult tell end start consider cellular phone data network cell phone voice datum communication cell phone give area connect radio wave cell tower contain receiver transmitter network similar lan cell phone communicate people talk exchange datum happen connect tower tower connect tower hub connect tower communication land line communication medium route packet destination network wan like appropriate tower receive packet use transmitter send correct recipient 19.3 communication structure discuss physical aspect networking turn naming resolution issue network communication involve naming system network process site exchange information process site b able specify computer system process process identifier message address process identifier networked system share memory host system initially knowledge process solve problem process remote system generally identify pair < host identifier > host unique network identifie process identifier unique number host ahost usually alphanumeric identifier number easy user specify instance site host name program student faculty cs host program certainly easy remember numeric host address 128.148.31.100 name convenient human use computer prefer number speed simplicity reason mechanism resolve host host id describe destination system net- work hardware mechanism similar address bind- ing occur program compilation linking loading execution chapter 9 case host name possibility exist host data file contain name numeric address host reachable network similar bind compile time problem model add remove host network require update data file host fact early day arpanet canonical host file copy system periodically network grow method untenable alternative distribute information system network network use protocol

method untenable alternative distribute information system network network use protocol distribute retrieve information scheme like execution time binding internet use domain system dns host resolution dns specify naming structure host address resolution host internet logically address multipart name know ip address part ip address progress specific general period separate field instance eric.cs.yale.edu refer host eric department computer science yale university level domain edu level domain include com

commercial site org organization domain country connect network system specify country organization type generally system resolve address examine host component reverse order component server simply process system accept return address server responsible final step server host question contact host id return example request process system communicate eric.cs.yale.edu result following step 1 system library kernel system issue request server edu domain ask address server yale.edu server edu domain know address query 2 edu server return address host yale.edu server reside 3 system query server address ask 4 address return finally request address eric.cs.yale.edu return internet address host id host protocol inefficient individual host cache ip address resolve speed process course content cache refresh time case server move networks distributed systems usage java dnslookup < ip > i.e. java dnslookup www.wiley.com public class dnslookup public static void main(string arg hostaddress = inetaddress.getbyname(args[0 catch unknownhostexception

uhe system.err.println("unknown host + args[0 java program illustrate dns lookup address change fact protocol important optimize time safeguard add consider happen primary edu server crash possible edu host able address resolve make unreachable solution use secondary backup server duplicate content primary server domain service introduce host internet need copy file mention contain name address host network change file register site host sri nic periodically host copy update file sri nic able contact new system find host address change domain service name- server site responsible update host information domain instance host change yale university responsibility server yale.edu need report dns lookup automatically retrieve update information contact yale.edu directly domain contain autonomous subdomain distribute responsibility host host id change java provide necessary api design program map ip name ip address program show figure 19.4 pass ip eric.cs.yale.edu command line output ip address host return message indicate host resolve inetaddress java class represent ip address static method getbyname belong inetaddress class pass string representation ip return correspond inetaddress program invoke gethostaddress method internally use dns look ip address designate host generally operate system responsible accept pro- cesses message destine < host identifier > transfer message appropriate host kernel destination host responsible transfer message process name identifier process describe section 19.3.4 design communication network deal inherent complexity coordinate asynchronous operation communicate potentially slow error prone environment addition system network agree protocol set protocol determin- e host name locate host network establish connection simplify design problem related implementation partition problem multiple layer layer system com- municate equivalent layer system typically layer protocol communication take place peer layer specific protocol protocol implement hardware software instance figure 19.5 show logical communication computer low level layer implement hardware international standards organization create open systems inter- connection osi model describe layer networking layer implement practice useful understand networking logically

describe layer networking layer implement practice useful understand networking logically work describe layer 1 physical layer physical layer responsible handle mechanical electrical detail physical transmission bit stream physical layer communicate system agree electrical representation binary 0 1 datum send stream electrical signal receiver able interpret datum real system environment computer communicate osi network model network distributed systems properly binary datum layer implement hardware network device responsible deliver bit layer 2 data link layer data link layer responsible handle frame fix length part packet include error detection recovery occur physical layer send frame physical layer 3 network layer network layer responsible break mes- sage packet provide connection logical address route packet communication network include handle address outgoing packet decode address incoming pack- et maintain routing information proper response change load level router work layer layer 4 transport layer transport layer responsible transfer message node maintain packet order control flow avoid congestion layer 5 session layer session layer responsible implement session process process communication protocol layer 6 presentation layer presentation layer responsible resolve difference format site net- work include character conversion half duplex duplex mode layer 7 application layer application layer responsible inter- act directly user layer deal file transfer remote login protocol electronic mail schema distribute figure 19.6 summarize osi protocol stack set cooperate pro- tocol show physical flow datum mention logically layer protocol stack communicate equivalent layer system physically message start application layer pass low level turn layer modify message include message header datum equivalent layer receiving ultimately message reach data network layer transfer packet figure 19.7 data link layer target system receive datum message move protocol stack analyze modify strip header progress finally reach application layer use receiving process osi model formalize early work network proto- cols develop late 1970s currently widespread use widely adopt protocol stack tcp ip

1970s currently widespread use widely adopt protocol stack tcp ip model some- time call internet model adopt virtually internet site tcp ip protocol stack few layer osi model theoreti- cally combine function layer difficult implement efficient osi networking relationship osi tcp ip model show figure 19.8 tcp ip application layer identify protocol widespread use internet include http ftp ssh dns smtp transport layer end end message transfer connection management error control fragmentation flow control physical connection network termination equipment network routing addressing setup clearing file transfer access management document message interchange job transfer manipulation end user application process dialog synchronization control application entity framing datum transparency error control mechanical electrical osi protocol stack identify unreliable connectionless user datagram protocol udp reliable connection orient transmission control protocol tcp inter- net protocol ip responsible route ip datagram packet internet tcp ip model formally identify link physical layer allow tcp ip traffic run physical network section 19.3.3 consider tcp ip model run ethernet network security concern design implementation modern communication protocol strong authentication encryption need secure communication strong authentication ensure sender receiver communication suppose encryption protect content communication eavesdrop- ping weak authentication clear text communication com- mon variety reason common protocol design security frequently important performance sim- networks distributed systems osi network message plicity efficiency legacy show today add security existing infrastructure prove difficult complex strong authentication require multistep handshake protocol authen- tication device add complexity protocol encryption require- ment modern cpu efficiently perform encryption frequently include cryptographic acceleration instruction system performance compro- mise long distance communication secure authenticate http dns telnet osi tcp ip protocol stack endpoint encrypt stream packet virtual private net- work discuss section 16.4.2 lan communication remain unencrypted site protocol nfs version 4 include strong native authentication encryption help improve lan security address resolution examine operation respect tcp ip

improve lan security address resolution examine operation respect tcp ip protocol stack internet consider processing need transfer packet host different ethernet network base description ipv4 protocol type commonly tcp ip network host associate ip address host id string unique space manage segment describe early hierarchical describe host organization host associate host id split network number host number proportion split vary depend size network internet administrator assign network number site number free assign host id send system check routing table locate router send frame way routing table configure manually system administrator populate routing protocol border gateway protocol bgp router use network host- d transfer packet source network destination network destination system receive packet packet complete message component message packet need message reassemble pass tcp udp transport layer transmission destination process network packet sender host router receiver ethernet device unique byte number call medium access control mac address assign addressing device lan communicate number system need send datum system network software generate address resolution protocol arp packet contain ip address destination system packet broadcast system ethernet network broadcast use special network address usually maximum address signal host receive process packet broadcast send router different network system local network receive system ip address match ip address arp request respond send mac address system initiate query efficiency host cache ip mac address pair internal table cache entry age entry eventually remove cache access system require give time way host remove network eventually forget add performance arp entry heavily host pin arp cache ethernet device announce host id address commu- nication begin process specify host communicate networking software take determine ip address target dns lookup entry local host file networks distributed systems preamble start packet start frame delimiter length data section 2 6 2 6 byte pattern 10101010 ethernet address broadcast length byte message >

63 byte long error detection ethernet packet translation manually store message pass application layer software layer hardware layer hardware layer packet ethernet address start trailer indicate end packet contain checksum detection packet damage figure 19.9 packet place network ethernet device data section packet contain datum

original message contain upper level header compose message word part original message send source destination header 802.3 layer data link layer include datum ethernet packet destination local network source system look arp cache find ethernet address host place packet wire destination ethernet device see address packet read packet pass protocol stack destination system network different source source system find appropriate router network send packet router pass packet wan reach destination network router connect destination network check arp cache find ethernet number destination send packet host transfer data link layer header change ethernet address router chain header packet remain packet receive process protocol stack finally pass receive process kernel transport protocol udp tcp host specific ip address receive packet pass correct wait process transport protocol tcp udp identify receive send process use port number host single ip address multiple server process run wait packet long server process specify different port number default common service use know port number example include ftp 21 ssh 22 smtp 25 http 80 exam- ple wish connect http website web browser browser automatically attempt connect port 80 server number 80 port number tcp transport header extensive list know port log favorite linux unix machine look file /etc service transport layer accomplish connect network packet run process desire add reliability network packet stream explain outline general behavior transport protocol udp tcp user datagram protocol transport protocol udp unreliable bare bone extension ip addition port number fact udp header simple contain field source port number destination port number length checksum packet send quickly destination udp guarantee delivery low layer network stack packet lose packet arrive receiver order application figure error case adjust

adjust figure 19.10 illustrate common scenario involve loss packet client server udp protocol note protocol know connectionless protocol connection setup beginning transmission set state client start

connectionless protocol connection setup beginning transmission set state client start send datum similarly connection teardown client begin send sort request information server server respond send datagram packet client unfortunately packet drop overwhelmed router client packet use logic program application request miss packet server start send datum client example udp datum transfer drop packet networks distributed systems need use different transport protocol want additional reliability guarantee handle network transmission control protocol tcp transport protocol reliable connection orient addi- tion specify port number identify send receive process different host tcp provide abstraction allow send process host send order uninterrupted byte stream network receive process host accomplish thing host send packet receiver send acknowledg- ment packet ack notify sender packet receive ack receive timer expire sender send tcp introduce sequence number tcp header packet number allow receiver 1 packet order send datum request process 2 aware packet miss byte stream tcp connection initiate series control packet sender receiver call way handshake close gracefully control packet responsible tear connec- tion control packet allow sender receiver set remove state figure 19.11 demonstrate possible exchange tcp connection setup tear omit connection establish client send request packet server sequence number 904 unlike server udp example server send ack packet client server start send stream datum packet start different sequence number client send ack packet data packet receive unfortunately data packet sequence number 127 lose ack packet send client sender time wait ack packet resend data packet 127 later connection server send data packet sequence number 128 ack lose server receive ack resend data packet 128 client receive duplicate packet client know previously receive packet sequence number throw duplicate away send ack server allow server continue actual tcp specification ack require packet instead receiver send cumulative ack ack series packet server send numerous data packet sequentially wait ack

series packet server send numerous data packet sequentially wait ack advantage network throughput tcp help regulate flow packet mechanism call flow control congestion control flow control involve prevent sender overrun capacity receiver example receiver network distributed operating systems server start send datum client data seq = 904 data seq = 126 data seq = 128

datum seq = 128 datum seq = 127 datum seq = 127 ack 904 ack 126 ack 127 ack 128 ack 128 example tcp datum transfer drop packet slow connection slow hardware component like slow network card processor flow control state return ack packet receiver alert sender slow speed congestion control attempt approximate state network generally router sender receiver router overwhelmed packet tend drop drop packet result ack timeout result packet saturate network prevent condition sender monitor connection drop packet notice packet acknowledge drop packet sender slow rate send help ensure tcp connection fair connection happen time utilize reliable transport protocol like tcp distribute system need extra logic deal lost order packet tcp slow udp 19.4 network distributed operating systems section describe general category network orient operating system network operate system distribute operate sys- networks distributed systems tem network operate system simple implement generally difficult user access use distribute operating system provide feature network operating systems network operate system provide environment user access remote resource implement resource sharing log appropriate remote machine transfer datum remote machine machine currently general purpose operate sys- tem embed operate system android ios network operate system important function network operate system allow user log remotely internet provide ssh facility purpose illustrate suppose user westminster college wish compute kristen.cs.yale.edu computer locate yale university user valid account machine log remotely user issue command command result formation encrypt socket connection ten.cs.yale.edu computer connection establish network software create transparent bidirectional link character enter user send process kristen.cs.yale.edu output process send user process remote machine ask user login password correct information receive process act proxy user compute remote machine local user remote file transfer major function network operate system provide mech- anism remote fil transfer machine envi- ronment computer maintain local file system user site kurt albion.edu

want access file own becca locate computer colby.edu file copy explicitly computer colby maine computer albion michigan communication directional individual user site wish transfer file sean colby.edu karen albion.edu likewise issue set command internet provide mechanism transfer file transfer protocol ftp private secure file transfer protocol sftp suppose user carla wesleyan.edu want copy file own owen kzoo.edu user invoke sftp program execute network distributed operating systems program ask user login password correct information receive user use series command upload file download file navigate remote file system structure command transfer file remote machine local machine transfer file local machine remote machine ls dir list file current directory remote machine cd change current directory remote machine command change transfer mode binary ascii file determine connection status basic cloud base storage application allow user transfer file ftp user upload file cloud server download file local computer share file cloud service user web link sharing mechanism graphical interface common example include dropbox google drive important point ssh ftp cloud base storage application

require user change paradigm ftp example require user know command set entirely different normal operating- system command ssh user know appropriate command remote system instance user windows machine connect remotely unix machine switch unix command duration ssh session networking session complete round communica- tion frequently begin login authenticate end logoff terminate communication cloud base storage application user log cloud service usually web browser native application use series graphical command upload download share file obviously user find convenient require use different set command distributed operating system design address problem distributed operating systems distribute operating system user access remote resource way access local resource datum process migration site control distribute operating system depend goal system implement datum migration computation migration process migration combination thereof suppose user site want access datum file reside site b. system transfer datum basic method approach datum migration transfer entire file site a. point access file local user long need access file copy file modify send site b. networks distributed systems modest change large file

datum transfer mechanism think automated ftp system approach andrew file system find inefficient approach transfer site portion file actually necessary immediate task portion require later transfer place user long want access file modify send site b. note similarity demand paging modern distribute system use whichever method datum migration include mere transfer datum site system perform var- ious datum translation site involve directly compatible instance use different character code representation represent integer different number order bit circumstance want transfer computation datum system process call computation migration example consider job need access large file reside different site obtain summary file efficient access file site reside return desire result site initiate computation generally time transfer datum long time execute remote command remote command computation carry different way suppose process p want access file site a. access file carry site initiate rpc rpc use network protocol execute routine remote system section 3.8.2 process p invoke predefine procedure site a. procedure execute appropriately return result p. alternatively process p send message site a. operate system site create new process q function carry designate task process q complete execution send need result p message system scheme process p execute concurrently process q.

fact process run concurrently site method access file chunk file reside site rpc result invocation rpc transfer message site similarly process q course execution send message site turn create process process send message q repeat cycle logical extension computation migration process migration process submit execution execute site initiate entire process part execute different site scheme reason design issue distribute system load balancing process subprocesse distribute site workload computation speedup single process divide number subprocesse run concurrently different site node total process turnaround time reduce hardware preference process characteristic suitable execution specialized processor matrix inversion gpu microprocessor software preference process require software available particular site software move expensive process datum access computation migration datum computation numerous efficient process run remotely server host large database transfer datum run process locally use complementary technique process computer network system attempt hide fact process migrate client client need code program explicitly accomplish migration method usually employ achieve load balancing computation speedup homogeneous system need user input help execute program remotely approach allow require user specify explicitly process migrate method usually employ process move satisfy hardware software preference probably realize world wide web aspect distribute computing environment certainly provide datum migra- tion web server web client provide computation migration instance web client trigger database operation web server finally java javascript similar language provide form process migration java applet javascript script send server client execute network operate system provide feature distribute operating system make seamless easily accessible result powerful easy use facility reason huge growth world wide web 19.5 design issue distribute system designer distribute system number design challenge account system robust withstand failure system transparent user term file location user mobility finally system scalable allow addition computation power storage user briefly introduce issue section context describe design specific distribute file system networks distributed systems distribute system design specific distribute file system networks distributed systems distribute system suffer type hardware failure failure link host site loss message common type ensure system robust detect failure reconfigure system computation continue recover failure repair asystem fault tolerant tolerate certain level failure continue function normally degree fault tolerance depend design distribute system specific fault obviously fault tolerance well use term fault tolerance broad sense communication fault certain machine failure storage device crash decay storage medium tolerate extent afault tolerant system continue function degraded form face failure degradation affect performance functionality pro- portional failure cause system grind halt component fail certainly fault tolerant unfortunately fault tolerance difficult expensive implement network layer multiple redundant communication path network device switch router need avoid communication failure storage failure cause loss operate system application datum storage unit include redundant hardware component auto- matically case failure addition raid system ensure continued access datum event storage device failure section 11.8 environment share memory generally differentiate link failure site failure host failure message loss usu- ally detect failure occur failure detect appropriate action take action appropriate depend particular application detect link site failure use heartbeat procedure suppose site b direct physical link fix interval site send message site adoe receive message predetermine time period assume site b fail link aand b fail message b lose point site choice wait time period receive message b send message b. time go site astill receive message site send message receive reply procedure repeat conclusion site draw safely type failure occur site try differentiate link failure site failure send- e message b route exist b receive message immediately reply positively positive reply tell b failure direct link know advance long message travel b use time scheme time send design issue distribute systems message specify time interval willing wait reply b. receive reply message

specify time interval willing wait reply b. receive reply message time interval safely conclude b time occur conclude follow situation site b direct link exist b alternative path b message lose use reliable transport pro- tocol tcp eliminate concern site determine event occur suppose site discover mechanism describe failure occur initiate procedure allow system reconfigure continue normal mode operation direct link b fail information broadcast site system routing table update system believe site fail site long reach site system notify long attempt use service fail site failure site serve central coordinator activity deadlock detection require election new coordinator note site fail reach undesirable situation site serve coordinator network partition coordinator partition initiate conflicting action example coordinator responsible implement mutual exclusion situation process execute simultaneously critical section recovery failure failed link site repair integrate system gracefully smoothly suppose link b fail repair b notify accomplish notification continu- ously repeat heartbeat procedure describe section 19.5.1.1 suppose site b fail recover notify site site b receive information site update local table example need routing- table information list site undelivered message networks distributed systems transaction log unexecuted transaction mail site fail simply reach need information make multiple processor storage device distribute system transparent user key challenge designer ide- ally distribute system look user like conventional cen- tralize system user interface transparent distribute system distinguish local remote resource user able access remote resource resource local distribute system responsible locate resource arrange appropriate interaction aspect transparency user mobility convenient allow user log machine system force use specific machine transparent distribute system facilitate user mobil- ity bring user environment example home directory log protocol like ldap provide authentication system local remote mobile user authentication complete facility like desktop virtualization allow user desktop session remote issue scalability capability system adapt increase service load system bound resource completely saturated

system adapt increase service load system bound resource completely saturated increase load example respect file system saturation occur server cpu run high utilization rate disk o request overwhelm o subsystem scalability relative property measure accurately scalable system react gracefully increase load nonscalable performance degrade moderately second resource reach saturated state later perfect design accommodate grow load add new resource solve problem generate additional indirect load resource example add machine distribute system clog network increase service load worse expand system expensive design modification scalable system potential grow problem distribute system ability scale gracefully special importance expand network add new machine interconnect network commonplace short scalable design withstand high service load accommodate growth user community allow simple integration add resource scalability relate fault tolerance discuss early heavily load component paralyzed behave like faulty component addi- tion shift load faulty component component backup saturate generally have spare resource essential ensur- e reliability handle peak load gracefully multi- ple resource distribute system represent inherent advantage give system great potential fault tolerance scalability distributed file systems inappropriate design obscure potential fault tolerance scalability consideration design demonstrate distribution control datum scalability relate efficient storage scheme example cloud storage provider use compression deduplication cut storage compression reduce size file exam- ple zip archive file generate file file execute zip command run lossless compression algorithm datum speci- fie lossless compression allow original datum perfectly reconstruct compress datum result file archive small uncompressed file restore file original state user run sort unzip command zip archive file deduplication seek lower datum storage requirement remove redundant datum technology instance datum store entire system datum own multiple user compression deduplication perform file level block level technique automatically build distribute system compress information user explicitly issue command save storage space possibly cut network communication cost add user 19.6 distributed file systems world wide

communication cost add user 19.6 distributed file systems world wide web predominant distribute system use today important popular use distribute computing distributed fil system dfs explain structure dfs need define term service server client dfs context service software entity run machine provide particular type function client server service software run single machine client process invoke service set operation form client interface low level interface define actual cross- machine interaction intermachine interface terminology file system provide file service client client interface file service form set primitive file operation create file delete file read file write file primary hardware component file server control set local secondary storage device usually hard disk solid state drive file store retrieve accord client dfs file system client server storage device dis- perse machine distribute system accordingly service activ- ity carry network instead single centralized data repository system frequently multiple independent storage device concrete configuration implementation dfs vary system system configuration server run dedicated machine machine server client distinctive feature dfs multiplicity autonomy client server system ideally dfs appear client conventional centralized file system client interface networks distributed systems dfs distinguish local remote file dfs locate file arrange transport datum transparent dfs like transparent distribute system mention early facilitate user mobility bring user environment example user home directory user log important performance measure dfs time need satisfy service request conventional system time consist storage access time small cpu processing time dfs remote access additional overhead associate distribute structure overhead include time deliver request server time response network client direction addition transfer information cpu overhead run communication protocol software performance dfs view dimension dfs transparency performance ideal dfs comparable conventional file system basic architecture dfs depend ultimate goal widely architectural model discuss client server model cluster base model main goal client server architecture allow transparent file sharing client

main goal client server architecture allow transparent file sharing client file store locally individual client machine distribute file system nfs openafs prime example nfs common unix base dfs version refer nfs version 3 note application need run parallel large data set high availability scalability cluster base model appropriate client server model know example google file system open source hdfs run hadoop framework client server dfs model figure 19.12 illustrate simple dfs client server model server store file metadata attach storage system server store different file client connect server network request access file dfs contact server know protocol nfs version 3 server client server dfs model distributed file systems responsible carry authentication check request file per- mission warrant deliver file request client client make change file client deliver change server hold master copy file client server version file keep consistent way minimize network traffic server workload extent possible network file system nfs protocol originally develop sun microsystems open protocol encourage early adoption different architecture system beginning focus nfs simple fast crash recovery face server failure implement goal nfs server design stateless track client access file thing open file descriptor file pointer mean client issue file operation read file operation idempotent face server crash idempotent describe operation issue return result case read operation client keep track state file pointer simply reissue operation server crash come online read nfs implementation section 15.8 andrew fil system openafs create carnegie mellon uni- versity focus scalability specifically researcher want design protocol allow server support client possible mean minimize request traffic server client request file file content download server store client local storage update file send server file close new version file send client file open comparison nfs chatty send block read write request server file client openafs nfs mean addition local file sys- tem word format

openafs nfs mean addition local file sys- tem word format hard drive partition nfs file system instead server format partition local file system choosing ext4 export share directory dfs client simply attach export directory file system tree way dfs separate responsibility local file system concentrate distribute task dfs client server model design suffer single point failure server crash computer clustering help resolve problem redundant component clustering method failure detect fail work component continue server operation addition server present bottleneck request datum metadata result problem scalability bandwidth cluster base dfs model datum o workload processing expand need dfs fault tolerant scalable large bottleneck tolerate system component failure expect cluster base architecture develop meet need figure 19.13 illustrate sample cluster base dfs model basic model present google file system gfs hadoop distributed networks distributed systems example cluster base dfs model fil system hdfs client connect network master metadata server datum server house chunk por- tion file metadata server keep mapping datum server hold chunk file traditional hierarchical mapping directory file file chunk store datum server replicate certain number time example time protect compo- nent failure fast access datum server contain replicate chunk fast access chunk obtain access file client contact metadata server metadata server return client identity datum server hold request file chunk client contact close datum server server receive file information different chunk file read write parallel store different datum server metadata server need contact entire process make metadata server likely performance bottleneck metadata server responsible redistribute balance file chunk datum server gfs release 2003 support large distribute data intensive appli- cation design gfs influence main observation hardware component failure norm exception routinely expect file store system large file change append new datum end file overwrite existing datum redesign application file system api increase system consistent fourth observation gfs export api

system api increase system consistent fourth observation gfs export api require application program api dfs naming transparency shortly develop gfs google develop modularized software layer call mapreduce sit gfs mapreduce allow developer carry large scale parallel computation easily utilize benefit low layer file system later hdfs hadoop framework include stackable module like mapreduce hdfs create base google work like gfs mapreduce hadoop support processing large data set distribute computing environment suggest early drive framework occur traditional system scale capacity performance need big data project reasonable price example big data project include crawl analyze social medium customer datum large amount scientific data point trend 19.7 dfs naming transparency naming mapping logical physical object instance user deal logical datum object represent file name system manipulate physical block datum store disk track usually user refer file textual map low level numerical identifier turn map disk block multilevel mapping provide user abstraction file hide detail disk file store transparent dfs new dimension add abstraction hide network file locate conventional file system range naming mapping address disk dfs range expand include specific machine disk file store go step concept treat file abstraction lead possibility fil replication give file mapping return set location file replica abstraction existence multiple copy location hide need differentiate related notion mapping 1 location transparency file reveal hint file physical storage location 2 location independence file need change file physical storage location change definition relate level naming discuss previously file different name different level user level textual name system level numerical identifier location independent naming scheme dynamic mapping map file different location different time location independence strong property location transparency practice current dfss provide static location transparent mapping user level name support fil migration change location file automatically provide location independence openafs networks distributed systems support location independence file mobility example hdfs

networks distributed systems support location independence file mobility example hdfs include file migration follow posix standard provide flexibility implementation interface hdfs keep track location datum hide information client dynamic location trans- parency allow underlie mechanism self tune example amazon s3 cloud storage facility provide block storage demand api place storage sees fit move datum necessary meet performance reliability capacity requirement aspect differentiate location independence static divorce datum location exhibit location independence pro- vide well abstraction file file denote file significant attribute content loca- tion location independent file view logical data container attach specific storage location static location transparency support file denote specific hide set physical disk block static location transparency provide user convenient way share datum user share remote file simply name file location- transparent manner file local dropbox cloud base storage solution work way location independence pro- mote share storage space data object file mobilize overall system wide storage space look like single virtual resource possible benefit ability balance utilization storage system location independence separate naming hierarchy storage-device hierarchy intercomputer structure contrast static location transparency name transparent easily expose correspondence component unit machine machine configure pattern similar naming structure configuration restrict architecture system unnecessarily conflict consideration server charge root directory example structure dictate naming hierarchy contradict decentralization guideline separation location complete client access file reside remote server system fact client diskless rely server provide file include operating- system kernel special protocol need boot sequence consider problem get kernel diskless workstation diskless workstation kernel use dfs code retrieve kernel instead special boot protocol store read memory rom client invoke enable networking retrieve special file kernel boot code fix location kernel copy network load dfs make operate system file available advantage diskless client include low cost client machine require disk great convenience operate system upgrade occur server need modify dfs

great convenience operate system upgrade occur server need modify dfs naming transparency disadvantage add complexity boot protocol performance loss result use network local disk main approach naming scheme dfs simple approach file identify combination host local guarantee unique system wide ibis instance file identify uniquely host local local unix like path internet url system use approach naming scheme location transparent location independent dfs structure collection isolate component unit entire conventional file system component unit remain isolated mean provide refer remote file consider scheme second approach popularize nfs nfs provide means attach remote directory local directory give appearance coherent directory tree early nfs version allow previously mount remote directory access transparently advent automount feature allow mount demand base table mount point file structure name component integrate support trans- parent sharing integration limited uniform machine attach different remote directory tree result structure versatile achieve total integration component file system approach single global structure span file system openafs provide single global namespace file directory export allow similar user experience different client machine ideally compose file system structure struc- ture conventional file system practice special file example unix device file machine specific binary directory goal difficult attain evaluate naming structure look administrative complexity complex difficult maintain structure nfs structure remote directory attach local directory tree result hierarchy highly unstructured server unavailable arbitrary set directory different machine unavailable addition separate accreditation mechanism control machine allow attach directory tree user able access remote directory tree client deny access implementation transparent naming require provision mapping file associate location mapping manageable aggregate set file component unit provide mapping component unit basis single file basis aggregation serve administrative purpose unix like system use hierarchical directory tree provide location mapping aggregate file recursively directory networks distributed systems enhance availability crucial mapping information use replication local caching note location independence mean

mapping information use replication local caching note location independence mean mapping change time replicate mapping make simple consistent update information impossible overcome obstacle introduce low level location independent file identifier openafs use approach textual file name map low level file identifier indicate component unit file belong iden- tifier location independent replicate cache freely invalidate migration component unit inevitable price need second level mapping map component unit location need simple consistent update mechanism implement unix like directory tree low level location independent identi- fier make hierarchy invariant component unit migration aspect change component unit location mapping common way implement low level identifier use structured name name bit string usually part identify component unit file belong second identi- fie particular file unit variant part possible invariant structured name individual part unique time context rest part obtain uniqueness time take care reuse use add sufficiently bit method openafs timestamp apollo domain way view process take location transparent system ibis add level abstraction produce location independent naming scheme 19.8 remote file access let consider user request access remote file server store file locate naming scheme actual data transfer place way achieve transfer remote service mechanism request access deliver server server machine perform access result forward user common way implement remote service rpc paradigm discuss chapter 3 direct analogy exist disk access method conventional file system remote service method dfs remote service method analogous perform disk access access request ensure reasonable performance remote service mechanism use form caching conventional file system rationale caching reduce disk o increase performance dfs goal reduce network traffic disk o.

follow discussion describe implementation caching dfs contrast basic basic caching scheme concept caching simple datum need satisfy access request cache copy datum bring server remote file access client system access perform cache copy idea retain recently access disk block cache repeat access information handle locally additional network traffic replacement policy example recently algorithm keep cache size bound direct correspondence exist access traffic server file identify master copy reside server machine copy part file scatter different cache cache copy modify change need reflect master copy preserve relevant consistency semantic problem keep cache copy consistent master file cache- consistency problem discuss section 19.8.4 dfs caching easily call network virtual memory act similarly demand- page virtual memory back store usually remote server local disk nfs allow swap space mount remotely actually implement virtual memory network result performance penalty granularity cache datum dfs vary block file entire file usually datum cache need satisfy single access access serve cache datum procedure like disk read ahead section 14.6.2 openafs cache file large chunk 64 kb system discuss support caching individual block drive client demand increase cache unit increase hit ratio increase miss penalty miss require datum transfer increase potential consistency problem select unit caching involve consider parameter network transfer unit rpc protocol service unit rpc protocol network transfer unit ethernet packet 1.5 kb large unit cache datum need disassemble delivery reassemble reception block size total cache size obviously importance block- caching scheme unix like system common block size 4 kb 8 kb large cache 1 mb large block size 8 kb beneficial small cache large block size beneficial result few block cache low hit ratio cache datum store disk main memory disk cache clear advantage main memory cache reliable modification cache datum lose crash cache keep volatile memory cache datum keep disk recovery need fetch main

volatile memory cache datum keep disk recovery need fetch main memory cache advantage main memory cache permit workstation diskless datum access quickly cache main memory disk technology move large expensive memory result performance speedup predict outweigh advantage disk cache networks distributed systems server cache speed disk o main memory regardless user cache locate use main memory cache user machine build single caching mechanism use server user remote access implementation think hybrid caching remote service nfs instance implementation base remote service augment client- server memory caching performance evaluate method evaluate degree method emphasize nfs protocol implementation provide disk caching openafs policy write modify datum block server master copy critical effect system performance reliability simple policy write datum disk soon place cache advantage write policy reliability little information lose client system crash policy require write access wait information send server cause poor write performance cache write equivalent remote service write access exploit caching read access alternative delay write policy know write caching delay update master copy modification write cache write server later time policy advantage write write cache write access complete quickly second datum overwrite write case update need write unfortunately delay write scheme introduce reliability problem unwritten datum lose user machine variation delay write policy differ modify datum block flush server alternative flush block eject client cache option result good performance block reside client cache long time write server compromise alternative write policy scan cache regular interval flush block modify recent scan unix scan local cache nfs use policy file datum write issue server cache flush write reach server disk consider complete nfs treat metadata directory datum file attribute datum differently metadata change issue synchronously server file structure loss directory structure corruption avoid client server crash variation delay write write datum server file close write close policy openafs case file open

server file close write close policy openafs case file open short period modify rarely policy significantly reduce network traffic addition write close policy require closing process delay file write final thoughts distributed file systems reduce performance advantage delay write file open long period modify frequently performance advantage policy delay write frequent flushing client machine face problem decide locally cache copy datum consistent master copy client machine determine cache datum date cache date copy datum allow access approach verify validity cache datum 1 client initiate approach client initiate validity check contact server check local datum consistent master copy frequency validity checking crux approach determine result consistency semantic range check access check access file file open basically access couple validity check delay compare access serve immediately cache alternatively check initiate fix time interval depend frequency validity check load network 2 server initiate approach server record client file part file cache server detect potential inconsis- tency react potential inconsistency occur dif- ferent client conflict mode cache file unix semantic section 15.7 implement resolve potential inconsistency hav- e server play active role server notify file open intend mode read write indicate open server act detect file open simultaneously conflict mode disable caching particular file actually disable caching result switch remote service mode operation cluster base dfs cache consistency issue compli- cat presence metadata server replicate file datum chunk datum server early example hdfs gfs compare difference hdfs allow append write operation random write single file writer gfs allow random write concurrent writer greatly complicate write consistency guarantee gfs simplify hdfs 19.9 final thought distributed file systems line dfs client server cluster base architecture blur nfs version 4.1 specification include protocol parallel version nfs call pnfs writing adoption slow networks distributed systems gfs hdfs large scale dfs export non posix api transparently map directory regular

scale dfs export non posix api transparently map directory regular user machine nfs openafs system access dfs need client code instal software layer rapidly develop allow nfs mount dfs attractive advantage scalability advantage cluster base dfs allow native operating system utility user access file directly dfs writing open source hdfs nfs gateway support nfs ver- sion 3 work proxy hdfs nfs server software hdfs currently support random write hdfs nfs gateway support capability mean file delete recreate scratch byte change commercial organi- zation researcher address problem build stackable framework allow stacking dfs parallel compute module mapreduce distribute database export file volume nfs type file system complex cluster base dfs complex client server dfs clustered file system cfs parallel file system pfs cfs typically run lan system important widely deserve mention cover detail common cfs include lustre gpfs cfs essentially treat n system store datum y system access datum single client server instance nfs example server naming separate nfs server generally provide different naming scheme cfs knit storage content storage device server uniform transparent space gpfs file system structure lustre use exist file system zfs file storage management learn distributed file system common use today provide file sharing lan cluster environment wan complexity implement system underestimate especially con- sidere dfs operating system independent widespread adoption provide availability good performance presence long distance commodity hardware failure frail networking increase user workload distribute system collection processor share mem- ory clock instead processor local memory processor communicate communica- tion line high speed bus internet processor distribute system vary size function distribute system provide user access system resource access share resource provide datum migration computation migration process migration access specify user implicitly supply operate system application protocol stack specify network layering model add information message ensure reach destination aname system dns translate host network

message ensure reach destination aname system dns translate host network address protocol arp need translate network number network device address ethernet address instance system locate separate network router need pass packet source network destination network transport protocol udp tcp direct packet wait process use unique system wide port number addition tcp protocol allow flow packet reliable connection- orient byte stream challenge overcome distribute system work correctly issue include naming node process system fault tolerance error recovery scalability scalability issue include handle increase load fault tolerant efficient storage scheme include possibility compression and/or deduplication dfs file service system client server storage device disperse site distribute system accordingly service activity carry network instead single cen- tralize data repository multiple independent storage device main type dfs model client server model cluster base model client server model allow transparent file sharing client cluster base model distribute file datum server build large scale parallel ideally dfs look client like conventional centralized file system conform exactly traditional file system interface posix multiplicity dispersion server storage device transparent transparent dfs facilitate client mobility bring client environment site client approach naming scheme dfs sim- plest approach file name combination host local guarantee unique system wide approach popularize nfs provide means attach remote directo- rie local directory give appearance coherent directory request access remote file usually handle complemen- tary method remote service request access deliver server server machine perform access result forward client caching datum need satisfy access request cache copy datum bring server client access perform cache copy problem keep cache copy consistent master file cache consistency problem networks distributed systems bad idea router pass broadcast packet discuss advantage disadvantage cache transla- tion computer locate remote domain formidable problem designer solve imple- ment network system quality transparency build robust distribute system know kind failure occur list possible type failure distribute system specify entry list applicable crucial know message send arrive destination safely answer yes explain answer appropriate example distribute system site b. consider site distinguish follow b go link b go b extremely overloaded response time 100 time long normal implication answer recovery distribute peterson davie 2012 kurose ross 2017 provide general overview computer network internet protocol describe comer 2000 coverage tcp ip find fall stevens 2011 stevens 1995 unix network programming describe thoroughly steven et al 2003 ethernet wifi standard speed evolve quickly current ieee 802.3 ethernet standard find http://standards.ieee.org/about/get/ 802/802.3.html current ieee 802.11 wireless lan standard find sun network file system nfs describe callaghan 2000 infor- mation openafs available http://www.openafs.org mawat et al 2003 google mapreduce method describe tributed file system discuss k. shvachko chansler 2010 hadoop framework discuss http://hadoop.apache.org/. learn lustre http://lustre.org b. callaghan nfs illustrated addison wesley 2000 d. comer internetworking tcp ip volume fourth edition prentice hall 2000 fall stevens 2011 k. fall r. stevens tcp ip illustrated volume 1 protocols second edition john wiley sons 2011 ghemawat et al 2003 s. ghemawat h. gobioff s.-t. leung google file system proceedings acm symposium operating systems k. shvachko chansler 2010 r. chansler hadoop distributed file system 2010 kurose ross 2017 j. kurose k. ross computer networking approach seventh edition addison wesley 2017 peterson davie 2012 l. l. peterson b. s. davie computer networks systems approacm fifth edition morgan kaufmann 2012 steven et al 2003 r. steven b. fenner

a. rudoff unix network program- ming volume 1 sockets networking api edition john wiley sons r. stevens tcp ip illustrated volume 2 implementation chapter 19 exercise difference computation migration process migration easy implement osi model networking specify seven layer functionality computer system use few layer implement network use few layer problem use few layer cause explain double speed system ethernet segment result decrease network performance udp transport protocol change help solve problem advantage dedicated hardware device router disadvantage device compare general purpose computer way server well static host table problem complication associate server method use decrease traffic server generate satisfy translation request server organize hierarchical manner pur- pose hierarchical organization low layer osi network model provide datagram service delivery guarantee message transport layer protocol tcp provide reliability discuss advantage disadvantage support reliable message delivery low run program show figure 19.4 determine ip address following host name dns map multiple server www.google.com run program show figure 19.4 ip address modify program display server ip address instead original http protocol tcp ip underlie network protocol page graphic applet separate tcp session construct tear overhead build destroy tcp ip connection performance problem result implementation method udp tcp good alternative change improve advantage disadvantage make com- puter network transparent user benefit dfs compare file system central- following workload identify cluster base client server dfs model handle workload well explain host student file university lab process datum send hubble telescope share datum multiple device home server discuss openafs nfs provide follow location transparency b location independence transparent dfs circumstance prefer location independent dfs discuss reason preference aspect distribute system select system run totally reliable network compare contrast technique cache disk block locally client system remotely server scheme likely result great space saving multiuser dfs file level deduplication block level deduplication explain answer type extra metadata

level deduplication block level deduplication explain answer type extra metadata information need store dfs use deduplication integrate concept describe early book examin- e real operate system cover system detail linux windows 10 choose linux reason popular freely available represent featured unix system give student operate system opportunity read modify real operating- system source code windows 10 student examine modern operate sys- tem design implementation drastically different unix operating system microsoft popular desk- operating system operate system mobile device windows 10 modern design feature look feel different early operating system produce microsoft c h p t e r linux system update robert love chapter present depth examination linux operating system examine complete real system concept discuss relate practice linux variant unix gain popularity decade power device small mobile phone large room- fill supercomputer chapter look history development linux cover user programmer interface linux present interface owe great deal unix tradition discuss design implementation interface linux rapidly evolve operate system chapter describe development linux 4.12 kernel release 2017 explore history unix operate system linux derive principle linux design base examine linux process thread model illustrate linux schedule thread provide interprocess communication look memory management linux explore linux implement file system manage o device 20.1 linux history linux look feel like unix system unix compat- ibility major design goal linux project linux young unix system development begin 1991 finnish university student linus torvalds begin create small self- contain kernel 80386 processor true 32 bit processor intel range pc compatible cpu linux system early development linux source code available free cost minimal distributional restriction internet result linux history collaboration developer world correspond exclusively internet initial kernel partially implement small subset unix system service linux system grow include functionality expect modern unix system early day linux development revolve largely central operating system kernel core privileged

linux development revolve largely central operating system kernel core privileged executive manage sys- tem resource interact directly computer hardware need kernel course produce operate system need distinction linux kernel complete linux system linux kernel original piece software develop scratch linux community linux system know today include multitude component write scratch bor- row development project create collabora- tion team basic linux system standard environment application user programming enforce standard mean manage available functionality linux mature need arise layer functionality linux system need meet linux distribution linux distribution include standard component linux system plus set administrative tool simplify initial installation subsequent upgrading linux manage installation removal package system mod- ern distribution typically include tool management file system creation management user account administration network web browser word processor linux kernel linux kernel release public version 0.01 date 14 1991 networking run 80386 compatible intel processor pc hardware extremely limited device driver support vir- tual memory subsystem fairly basic include support memory map file early incarnation support share page copy write protect address space file system support minix file system linux kernel cross- developed minix platform milestone linux 1.0 release march 14 1994 release culminate year rapid development linux kernel single big new feature networking 1.0 include support unix standard tcp ip networking protocol bsd compatible socket interface networking programming device driver support

add run ip ethernet ppp slip protocol serial line 1.0 kernel include new enhanced file system limitation original minix file system support range scsi controller high performance disk access developer extend vir- tual memory subsystem support paging swap file memory mapping arbitrary file read memory mapping implement range extra hardware support include release restrict intel pc platform hardware support grow include floppy disk cd rom device sound card range mouse international keyboard floating point emulation provide kernel 80386 user 80387 math coprocessor system v unix style interprocess communication ipc include share memory semaphore message queue implement point development start 1.1 kernel stream numerous bug fix patch release subsequently 1.0 pattern adopt standard numbering convention linux kernel kernel odd minor version number 1.1 2.5 development kernel even- number minor version number stable production kernel update stable kernel intend remedial version development kernel include new relatively untested functionality pattern remain effect version 3 march 1995 1.2 kernel release release offer nearly improvement functionality 1.0 release support wide variety hardware include new pci hardware bus archi- tecture developer add pc specific feature support 80386 cpu virtual 8086 mode allow emulation dos operating system pc computer update ip implementation support accounting firewalling simple support dynamically loadable unloadable kernel module supply 1.2 kernel final pc linux kernel source distribution linux 1.2 include partially implement support sparc alpha mips cpu integration architecture begin stable 1.2 kernel release linux 1.2 release concentrate wide hardware support complete implementation exist functionality new functionality development time integration new code main kernel source code defer stable 1.2 kernel release result 1.3 development stream see great deal new functionality add kernel work release june 1996

linux version 2.0 release give major version number increment major new capabili- tie support multiple architecture include 64 bit native alpha port symmetric multiprocessing smp support additionally memory- management code substantially improve provide unified cache file system datum independent caching block device result change kernel offer greatly increase file system virtual- memory performance time file system caching extend networked file system writable memory map region support major improvement include addition internal ker- nel thread mechanism expose dependency loadable module support automatic loading module demand file system quota posix compatible real time process scheduling class linux system improvements continue release linux 2.2 1999 port ultrasparc system add networking enhance flexible firewalling improved routing traffic management support tcp large window selective acknowledgement acorn apple nt disk read nfs enhance new kernel mode nfs daemon signal handling interrupt o lock fine level improve symmetric multiprocessor smp performance advance 2.4 2.6 release kernel include increased support smp system journaling file system enhancement memory management block o system thread scheduler mod- ifie version 2.6 provide efficient o(1 scheduling algorithm addi- tion 2.6 kernel preemptive allow thread preempt run kernel mode linux kernel version 3.0 release july 2011 major version bump 2 3 occur commemorate twentieth anniversary linux new feature include improved virtualization support new page write facility improvement memory management system new thread scheduler completely fair scheduler cfs linux kernel version 4.0 release april 2015 time major version bump entirely arbitrary linux kernel developer simply grow tired large minor version today linux kernel version sig- nify release ordering 4.0 kernel series provide sup- port new architecture improved mobile functionality iterative improvement focus new kernel remainder chapter linux system note early linux kernel form core linux project component complete linux operate system linux kernel compose entirely code write scratch specifically linux project support software make linux

code write scratch specifically linux project support software make linux system exclusive linux common number unix- like operate system particular linux use tool develop berkeley bsd operate system mit x window system free software foundation gnu project sharing tool work direction main system library linux originate gnu project linux commu- nity greatly improve library address omission inefficiency bug component gnu c compiler gcc sufficiently high quality directly linux network administra- tion tool linux derive code develop 4.3 bsd recent bsd derivative freebsd borrow code linux return example sharing include intel float point emulation math library pc sound hardware device driver linux system maintain loose network develop- er collaborate internet small group individual have responsibility maintain integrity specific component small number public internet file transfer protocol ftp archive site act de facto standard repository component file system hierarchy standard document maintain linux community means ensure compatibility system component stan- dard specify overall layout standard linux file system determine directory name configuration file library system binary run time datum file store theory anybody install linux system fetch late revision necessary system component ftp site compile linux early day precisely linux user linux mature individual group attempt job painful provide standard precompile set package collection distribution include basic linux system typically include extra system installation manage- ment utility precompile ready install package common unix tool news server web browser text processing editing tool game distribution manage package simply provide mean unpack file appropriate place impor- tant contribution modern distribution advanced package management today linux distribution include package track database allow package instal upgrade remove painlessly sls distribution date early day linux collection linux package recognizable complete distribu- tion instal single entity sls lack package- management tool expect linux distribution slackware dis- tribution represent great improvement overall quality poor package management fact widely instal

great improvement overall quality poor package management fact widely instal distribution linux community slackware release commercial noncommercial linux distribution available red hat debian particularly pop- ular distribution come commercial linux support company second free software linux community commercially support version linux include distribution canonical suse linux distribution circulation list variety distribution prevent linux distribution compatible rpm package file format understand majority distribution commer- cial application distribute format instal run distribution accept rpm file linux kernel distribute version 2.0 gnu general public license gpl term set free software foundation linux public domain software public domain imply author code hold code author linux free software how- linux system sense people copy modify use manner want away sell copy main implication linux licensing term linux create derivative linux legitimate exercise distribute derivative include source code software release gpl redistribute binary product release software include component cover gpl gpl source code available alongside binary distribution restriction prohibit make sell binary software distri- bution long anybody receive binary give opportunity originate source code reasonable distribution charge

20.2 design principle overall design linux resemble traditional nonmicrokernel unix implementation multiuser preemptively multitaske system set unix compatible tool linux file system adhere traditional unix semantic standard unix networking model fully imple- mente internal detail linux design influence heavily history operate system development linux run wide variety platform originally develop exclusively pc architecture great deal early develop- ment carry

individual enthusiast fund development research facility start linux attempt squeeze functionality possible limited resource today linux run happily multiprocessor machine hundred gigabyte main memory terabyte disk space capable operate usefully 16 mb ram pc powerful memory hard disk cheap original minimalist linux kernel grow implement unix functionality speed efficiency important design goal recent current work linux concentrate major design goal standardization price pay diversity unix imple- mentation currently available source code write necessarily compile run correctly system call present different unix system necessarily behave exactly way posix standard comprise set specification different aspect operating system behavior posix document common operating system functionality extension pro- cess thread real time operation linux design comply relevant posix document linux distribution achieve official posix certification give standard interface programmer user linux present surprise anybody familiar unix detail interface section programmer interface section c.3 user interface section c.4 bsd apply equally linux default linux programming interface adhere svr4 unix semantic bsd behavior aseparate set library available implement bsd semantic place behavior differ significantly standard exist unix world certification linux respect standard slow certifica- tion available fee expense involve certify operate system compliance standard substantial support wide base application important operate system implementation standard major goal linux development formal certification addition basic posix standard linux currently support posix threading extension pthread subset posix extension real time process control component linux system linux system compose main body code line traditional unix implementation 1 kernel kernel responsible maintain important abstraction operate system include thing virtual memory process 2 system library system library define standard set function application interact kernel function implement operate system functionality need privilege kernel code important system library c library know libc addition provide standard c library libc implement user mode linux system interface critical system level interface 3 system utility system utility program perform

system level interface 3 system utility system utility program perform indi- vidual specialized management task system utility invoke initialize configure aspect system know daemon unix terminology run permanently handle task respond incoming network connection accept logon request terminal update log file figure 20.1 illustrate component linux system important distinction kernel every- thing kernel code execute processor privileged mode system share library loadable kernel module component linux system linux system access physical resource computer linux refer privileged mode kernel mode linux user code build kernel operating system support code need run kernel mode place system library run user mode unlike kernel mode user mode access control subset system modern operate system adopt message- pass architecture kernel internal linux retain unix historical model kernel create single monolithic binary main reason performance kernel code datum structure keep sin- gle address space context switch necessary thread call operate system function hardware interrupt deliver more- kernel pass datum request subsystem relatively cheap c function invocation complicated inter- process communication ipc single address space contain core scheduling virtual memory code kernel code include device driver file system networking code kernel component share melting pot room modularity way user application load share library run time pull need piece code linux kernel load unload module dynamically run time kernel need know advance module load truly independent loadable component linux kernel form core linux operate system provide functionality necessary manage process run thread provide system service arbitrate protect access hardware resource kernel implement feature require qualify oper- ating system operate system provide linux kernel complete unix system lack functionality behavior unix feature provide necessarily format unix application expect appear operating-system interface visible run application maintain directly kernel application call system library turn operate system service necessary system library provide type functionality simple level allow application system call linux

provide type functionality simple level allow application system call linux kernel make system involve transfer control unprivileged user mode privileged kernel mode detail transfer vary architecture architecture library care collect system argument necessary arrange argument special form necessary system library provide complex version basic system call example c language buffered file handle function implement system library provide advanced control file o basic kernel system call library provide routine correspond system call sort algorithm mathemat- ical function string manipulation routine function necessary support running unix posix application implement linux system include wide variety user mode program system utility user utility system utility include program necessary initialize administer system set networking interface add remove user system user util- itie necessary basic operation system require elevated privilege run include simple file management utility copy file create directory edit text file important user utility shell standard command line interface unix system linux support shell common bourne- shell bash 20.3 kernel module linux kernel ability load unload arbitrary section kernel code demand loadable kernel module run privileged kernel mode consequence access hardware capability machine run theory restriction kernel module allow thing kernel module implement device driver file system networking protocol kernel module convenient reason linux source code free anybody want write kernel code able compile modify kernel reboot new functionality recompile relink- e reload entire kernel cumbersome cycle undertake develop new driver use kernel module new kernel test new driver driver compile load run kernel course new driver write distribute module user benefit have rebuild kernel point implication cover gpl license linux kernel release proprietary component add new component release gpl source code available demand kernel module interface allow party write distribute term device driver file system distribute gpl kernel module allow linux system set standard minimal kernel extra device driver build device driver user

standard minimal kernel extra device driver build device driver user need load explicitly system startup load automatically system demand unload use example mouse driver load usb mouse plug system unload mouse unplug module support linux component 1 module management system allow module load memory communicate rest kernel 2 module loader unloader user mode utility work module management system load module memory linux system 3 driver registration system allow module tell rest kernel new driver available 4 conflict resolutio reserve hardware resource protect resource accidental use driver load module require load binary content ker- nel memory system sure reference module make kernel symbol entry point update point correct loca- tion kernel address space linux deal reference updating split job module loading separate section management section module code kernel memory handling symbol module allow reference linux maintain internal symbol table kernel symbol table contain set symbol define kernel compilation symbol explicitly export set export symbol constitute define interface module interact kernel export symbol kernel function require explicit request programmer special effort need import symbol module module writer use standard external linking c language external symbol reference module declare simply mark unresolved final module binary produce compiler module load kernel system utility scan module unresolved reference symbol need resolve look kernel symbol table correct address symbol currently run kernel substitute module code module pass kernel loading system utility resolve reference module look kernel symbol table module reject loading module perform stage module- loader utility ask kernel reserve continuous area virtual kernel memory module kernel return address memory allo- cat loader utility use address relocate module machine code correct loading address second system pass module plus symbol table new module want export kernel module copy verbatim previously allo- cat space kernel symbol table update new symbol possible use module load final module management component new symbol possible use module load final module management component module requester kernel define communication interface module management program connect connection establish kernel inform management process process request device driver file system network service currently load manager opportunity load service original service request complete module load manager process regularly query kernel dynamically load module use unload module long actively need module load remain isolated region memory let rest kernel know new functionality provide kernel maintain dynamic table know driver provide set routine allow driver add remove table time kernel make sure call module startup routine module load call module cleanup routine module unload routine responsible register module register type functionality limit type example device driver want register sep- arate mechanism access device registration table include following item device driver driver include character device print- er terminal mouse block device include disk drive network interface device file system file system implement linux virtual file system calling routine implement format store file disk equally network file system nfs virtual file system content generate demand linux /proc file system network protocol amodule implement entire networking proto- col tcp simply new set packet filter rule network binary format format specify way recognize load execute new type executable file addition module register new set entry sysctl /proc table allow module configure dynamically section 20.7.4 commercial unix implementation usually sell run vendor hardware advantage single supplier solution software vendor good idea hardware configuration possible pc hardware come vast number configuration large num- ber possible driver device network card video display adapter problem manage hardware configuration severe modular device driver support currently active set device dynamically variable linux provide central conflict resolution mechanism help arbitrate access certain hardware resource aim follow prevent module clash access hardware resource prevent autoprobe device driver probe auto detect device con- figuration prevent autoprobe device driver probe auto detect device con- figuration interfere exist device driver linux system resolve conflict multiple driver try access hardware example parallel printer driver parallel line ip plip network driver try talk parallel port end kernel maintain list allocate hardware resource pc limited number possible o port address hardware o address space interrupt line dma channel device driver want access resource expect reserve resource kernel database requirement incidentally allow system admin- istrator determine exactly resource allocate driver give point module expect use mechanism reserve advance hardware resource expect use reservation reject resource present use module decide proceed fail initialization attempt request unload continue carry alternative 20.4 process management process basic context user request activity service operate system compatible unix system linux use process model similar version unix linux operate differently unix key place section review traditional unix process model section c.3.2 introduce linux threading model fork exec process model basic principle unix process management separate step operation usually combine creation new process running new program new process create fork system new program run exec distinctly separate function create new process fork run new program new subprocess simply continue execute exactly program exactly point parent process run way run new program require new process create process exec time new binary object load process address space new executable start execute context exist process model advantage great simplicity necessary specify detail environment new program system run program new program simply run exist environment parent process wish modify environment new program run fork run original executable child process system call require modify child process finally execute new program unix process encompass information operate system maintain track context single execution single program linux break context number specific section broadly process property fall group process identity environment context process identity consist mainly following

group process identity environment context process identity consist mainly following item process id pid process unique identifier pid specify process operate system application make system signal modify wait process additional identifier associate process process group typically tree process fork single user command login session credential process associate user id group id user group discuss section 13.4.2 determine right process access system resource file personality process personality traditionally find unix sys- tem linux process associate personality identifier slightly modify semantic certain system call personality primarily emulation library request system call compatible certain variety unix namespace process associate specific view file- system hierarchy call namespace process share com- mon namespace operate share file system hierarchy pro- cesse child different namespace unique file system hierarchy root directory set mount file system identifier limited control process process group session identifier change process want start new group session credential change subject appro- priate security check primary pid process unchangeable uniquely identify process termination process environment inherit parent compose null terminated vector argument vector environment vector argument vector simply list command line argument invoke run program conventionally start program environment vector list = value pair associate name environment variable arbitrary textual value environment hold kernel memory store process user mode address space datum process stack argument environment vector alter new pro- cess create new child process inherit environment par- ent completely new environment set new program invoke call exec process supply environment new program kernel pass environment variable linux system program replace process current environment kernel leave environment command line vector interpretation leave entirely user mode library application passing environment variable process inheriting variable child process provide flexible way pass information component user mode system software important environment variable conventional meaning related part system software example term variable set type terminal connect user login session program

term variable set type terminal connect user login session program use variable determine perform operation user display move cursor scroll region text program multilingual support use lang variable determine language display system message program include multilingual support environment variable mechanism custom tailor operate system process basis user choose language select editor independently process identity environment property usually set process create change process exit process choose change aspect identity need alter environment contrast process context state run program time change constantly process context include scheduling context important process context scheduling context information scheduler need suspend restart process information include save copy process register float point register store separately restore need process use float point arithmetic incur overhead save state scheduling context include information schedule priority outstanding signal wait deliver process key scheduling context process kernel stack separate area kernel memory reserve use kernel mode code system call interrupt occur process execute use stack accounting kernel maintain accounting information resource currently consume process total resource consume process entire lifetime far file table file table array pointer kernel file structure represent open file make file o system call process refer file integer know fil descriptor fd kernel use index table file system context file table list exist open file file system context apply request open new file file system context include process root directory current working directory signal handler table unix system deliver asynchronous signal process response external event signal handler table define action response specific signal valid action include ignore signal terminate process invoke rou- tine process address space virtual memory context virtual memory context describe content process private address space discuss section 20.6 processes threads linux provide fork system duplicate process load new executable image linux provide ability create thread clone system linux distinguish pro- cesses thread fact linux generally use

system linux distinguish pro- cesses thread fact linux generally use term task process thread refer flow control program clone system behave identically fork accept argument set flag dictate resource share parent child process create fork share resource parent flag include file system information share memory space share signal handler share set open file share clone pass flag clone fs clone vm clone sighand clone file parent child task share file system information current work directory memory space signal handler set open file clone fashion equivalent create thread system parent task share resource child task flag set clone invoke associate resource share result functionality similar fork system lack distinction process thread possible linux hold process entire context main process datum structure hold context independent subcontext process file system context file descriptor table signal handler table virtual memory context hold separate datum structure process datum structure simply contain pointer structure number process easily share subcontext point subcontext incremente reference count argument clone system tell subcontext copy share new process give new identity new scheduling context essential linux process accord argument pass kernel create new subcontext datum structure initialize copy parent set new process use subcontext datum structure parent linux system fork system special case clone copy subcontext share scheduling job allocate cpu time different task operat- e system linux like unix system support preemptive multitasking system process scheduler decide thread run make decision way balance fairness performance different workload complicated challenge modern normally think scheduling running interrupting user thread aspect scheduling important linux run- ning kernel task kernel task encompass task request run thread task execute internally behalf kernel task spawn linux o subsystem linux separate process scheduling algorithm time sharing algorithm fair preemptive scheduling multiple thread design real time task absolute priority important scheduling algorithm routine time sharing task receive major

priority important scheduling algorithm routine time sharing task receive major overhaul version 2.6 kernel early version run variation traditional unix scheduling algorithm algorithm provide adequate support smp system scale number task system grow maintain fairness interactive task par- ticularly system desktop mobile device thread scheduler overhaul version

2.5 kernel version 2.5 implement scheduling algorithm select task run constant time know o(1)—regardless number task processor system new scheduler provide increase support smp include processor affinity load balancing change improve scalability improve interactive performance fairness fact prob- lem bad certain workload consequently thread scheduler overhaul second time linux kernel version 2.6 version usher completely fair scheduler cfs linux scheduler preemptive priority base algorithm separate priority range real time range 0 99 nice value range 20 19 small nice value indicate high priority increase nice value decrease priority nice rest system cfs significant departure traditional unix process scheduler core variable scheduling algorithm priority time slice time slice length time slice processor thread afford traditional unix system process fix time slice boost penalty high- low priority process respectively process run length time slice higher- priority process run low priority process simple algorithm non unix system employ simplicity work early time sharing system prove incapable deliver good interactive performance fairness today modern desktop mobile device cfs introduce new scheduling algorithm call fair scheduling eliminate time slice traditional sense instead time slice thread allot proportion processor time cfs calculate long thread run function total number runnable thread start cfs say n runnable thread afford 1n processor time cfs adjust allotment weight thread allotment nice value thread default nice value weight 1 priority unchanged thread small nice value high priority receive high weight thread large nice value low priority receive low weight cfs run thread time slice proportional process weight divide total weight runnable process calculate actual

proportional process weight divide total weight runnable process calculate actual length time thread run cfs rely config- urable variable call target latency interval time runnable task run example

assume target latency 10 millisecond assume runnable thread priority thread weight receive proportion processor time case target latency 10 millisecond process run 5 millisecond process run 5 millisecond process run 5 millisecond forth 10 runnable thread cfs run millisecond repeat 1,000 thread thread run 1 microsecond follow procedure describe switching cost schedule thread short length time inefficient cfs con- sequently rely second configurable variable minimum granularity minimum length time thread allot processor thread regardless target latency run minimum granularity manner cfs ensure switching cost grow unac- ceptably large number runnable thread increase significantly ber runnable thread remain reasonable fairness switching cost maximize switch fair scheduling cfs behave differently traditional unix process scheduler way notably see cfs eliminate concept static time slice instead thread receive proportion processor time long allotment depend thread runnable approach solve problem map priority time slice inherent preemptive priority base scheduling algorithm possible course solve problem way abandon classic unix scheduler cfs solve problem simple algorithm perform interactive workload mobile device compromise throughput performance large server linux system linux real time scheduling algorithm significantly simple fair scheduling employ standard time sharing thread linux implement real time scheduling class require posix.1b come first- serve fcfs round robin section 5.3.1 section 5.3.3 respectively case thread priority addition scheduling class scheduler run thread high priority thread equal priority run thread wait long differ- ence fcfs round robin scheduling fcfs thread continue run exit block round robin thread preempt move end scheduling queue round robin thread equal priority automatically time share linux real time scheduling soft hard real time scheduler offer strict guarantee relative priority real time thread kernel offer guarantee quickly real- time thread schedule thread runnable contrast hard real time system guarantee minimum latency thread runnable actually run way kernel schedule operation fundamentally different way schedule thread request kernel

kernel schedule operation fundamentally different way schedule thread request kernel mode execution occur way run program request operate system service explicitly system implicitly example page fault occur alternatively device controller deliver hardware interrupt cause cpu start execute kernel define handler problem kernel task try access internal datum structure kernel task middle access datum structure interrupt service routine execute service routine access modify datum risk datum corruption fact relate idea critical section portion code access share datum allow execute concurrently result kernel synchronization involve thread scheduling framework require allow kernel task run violate integrity share datum prior version 2.6 linux nonpreemptive kernel mean thread run kernel mode preempt higher- priority thread available run version 2.6 linux kernel fully preemptive task preempt run kernel linux kernel provide spinlock semaphore reader writer version lock lock kernel smp machine fundamental locking mechanism spinlock kernel design spinlock hold short duration single processor machine spinlock appropriate use replace enable dis- able kernel preemption hold spinlock task dis- able kernel preemption task release spinlock enable kernel preemption pattern summarize acquire spin lock release spin lock disable kernel preemption enable kernel preemption linux use interesting approach disable enable kernel preemp- tion provide simple kernel interface preempt disable pre- empt enable addition kernel preemptible kernel mode task hold spinlock enforce rule task system thread info structure include field preempt count counter indicate number lock hold task counter incremente lock acquire decremente lock release value preempt count task currently run great zero safe preempt kernel task currently hold lock count zero kernel safely interrupt assume outstanding call preempt disable spinlock enabling disabling kernel preemption kernel lock hold short duration lock hold long period semaphore second protection technique linux apply critical sec- tion occur interrupt service routine basic tool processor interrupt control hardware disable interrupt spinlock critical section kernel guarantee proceed

control hardware disable interrupt spinlock critical section kernel guarantee proceed risk concurrent access shared datum structure penalty disable interrupt hardware architecture interrupt enable disable instruction cheap importantly long interrupt remain disabled o suspend device wait service wait interrupt reen- able performance degrade address problem linux kernel use synchronization architecture allow long critical section run entire duration have interrupt disable ability espe- cially useful networking code interrupt network device driver signal arrival entire network packet result great deal code execute disassemble route forward packet interrupt service routine linux implement architecture separate interrupt service routine section half half half standard interrupt service routine run recursive interrupt disable inter- rupt number line disable interrupt run half service routine run interrupt enable miniature scheduler ensure half interrupt them- self half scheduler invoke automatically inter- rupt service routine exit separation mean kernel complete complex process- ing response interrupt worry interrupt interrupt occur half exe- linux system cuting interrupt request half execute execution defer currently run complete execution half interrupt half interrupt similar half half half architecture complete mechanism disable select half execute normal foreground kernel code kernel code critical section easily system interrupt handler code critical section half fore- ground kernel want enter critical section disable relevant half prevent critical section interrupt end critical section kernel reenable half run half task queue half interrupt service routine critical section figure 20.2 summarize level interrupt protection kernel level interrupt code run high level interrupt code run low level user mode code user thread preempt thread time sharing scheduling interrupt occur linux 2.0 kernel stable linux kernel support symmetric multiprocessor smp hardware allow separate thread execute par- allel separate processor original implementation smp impose restriction processor time execute kernel code version 2.2 kernel single kernel spinlock term bkl big kernel lock create allow multiple thread run different processor

big kernel lock create allow multiple thread run different processor active kernel concurrently bkl provide coarse level locking granularity result poor scalability machine processor thread later release kernel smp implementation scalable split single kernel spinlock multiple lock protect small subset kernel data structure spinlock describe section 20.5.3 3.0 4.0 kernel provide additional smp enhancement include fine locking processor affinity load balance algorithm support hundred thousand physical processor single system half interrupt handler half interrupt handler kernel system service routine preemptible user mode program preemptible interrupt protection level 20.6

memory management memory management linux component deal allocate free physical memory page group page small block ram second handle virtual memory memory map address space run process section describe component examine mechanism loadable component new program bring process virtual memory response exec system management physical memory specific hardware constraint linux separate physical memory different zone region zone dma zone dma32 zone normal zone highmem zone architecture specific example intel x86 32 architecture certain isa industry standard architecture device access low 16 mb physical memory dma system 16 mb physical memory comprise zone dma system certain device access 4 gb physical memory despite support 64 bit address system 4 gb physical memory comprise zone dma32 zone highmem high memory refer physical memory map kernel address space example 32 bit intel architecture 232 provide 4 gb address space kernel map 896 mb address space remain memory refer high memory allocate zone highmem finally zone normal comprise normal regularly map page architecture give zone depend constraint modern 64 bit architecture intel x86 64 small 16 mb zone dma legacy device rest memory zone normal relationship zone physical address intel x86 32 archi- tecture show figure 20.3 kernel maintain list free page < 16 mb 16 896 mb > 896 mb relationship zone physical address intel x86 32 linux system zone request physical memory arrive kernel satisfy request appropriate zone primary physical memory manager linux kernel page allocator zone allocator responsible allocate free physical page zone capable allocate range physically contiguous page request allocator use buddy system section 10.8.1 track available physical page scheme adja- cent unit allocatable memory pair allocatable memory region adjacent partner buddy allocate partner region free combine form large region buddy heap large region partner combine form large free region conversely small memory request satisfy allocation exist small free region

conversely small memory request satisfy allocation exist small free region large free region subdivide partner satisfy request separate link list record free memory region allowable size linux small size allocatable mechanism single physical page figure 20.4

show example buddy heap allocation 4 kb region allocate small available region 16 kb region break recursively piece desire size available ultimately memory allocation linux kernel stat- ically driver reserve contiguous area memory system boot time dynamically page allocator

kernel function use basic allocator reserve memory specialized memory- management subsystem use underlie page allocator manage pool memory important virtual memory system describe section 20.6.2 kmalloc variable length allocator slab allocator allocate memory kernel datum structure page cache cache page belong file component linux operate system need allocate entire page request small block memory require kernel provide additional allocator arbitrary sized request size request know advance byte analo- gous c language malloc function kmalloc service allocate entire physical page demand split small piece splitting memory buddy system slab allocator linux kernel maintain list page use kmalloc service allocate memory involve determine appropriate list take free piece available list allocate new page split memory region claim kmalloc system allocate permanently free explicitly correspond kfree kmal- loc system reallocate reclaim region response memory strategy adopt linux allocate kernel memory know slab allocation slab allocate memory kernel datum struc- ture physically contiguous page cache consist slab single cache unique kernel datum structure example cache data structure represent pro- cess descriptor cache file object cache inode forth cache populate object instantiation kernel datum struc- ture cache represent example cache represent inode store instance inode structure cache represent process descriptor store instance process descriptor structure relationship slab cache object show figure 20.5 figure show kernel object 3 kb size object 7 kb size object store respective cache 3 kb 7 kb object slab allocation algorithm use cache store kernel object cache create number object allocate cache number object cache depend size associate slab example 12 kb slab contiguous 4 kb page store 2 kb object initially object cache mark free new object kernel datum structure need allocator assign free object cache satisfy request object assign cache mark let consider scenario kernel request memory slab allocator object represent process descriptor linux sys- tem process descriptor type

object represent process descriptor linux sys- tem process descriptor type struct task struct require linux system approximately 1.7 kb memory linux kernel create new task request necessary memory struct task struct object cache cache fulfill request struct task struct object allocate slab mark free linux slab possible state 1 object slab mark 2 object slab mark free 3 partial slab consist free object slab allocator attempt satisfy request free object partial slab exist free object assign slab slab available new slab allocate contiguous physical page assign cache memory object allocate main subsystem linux management physical page page cache virtual memory system system closely relate page cache kernel main cache file main mechanism o block device section 20.8.1 perform file system type include native linux disk base file system nfs networked file system perform o page cache page cache store entire page file content limit block device cache networked datum virtual memory system manage content process virtual address space system interact closely read page datum page cache require map page page cache virtual memory system follow section look virtual memory system great detail linux virtual memory system responsible maintain address space accessible process create page virtual memory demand manage load page disk swap disk require linux virtual memory manager maintain separate view process address space set separate region set page view address space logical view describe instruc- tion virtual memory system receive concern layout address space view address space consist set nonoverlap- ping region region represent continuous page align subset address space region describe internally single vm area struct structure define property region include process read write execute permission region information file associate region region address space link balanced binary tree allow fast lookup region correspond virtual address kernel maintain second physical view address space view store hardware page table process

physical view address space view store hardware page table process page- table entry identify exact current location page virtual memory disk physical memory physical view manage set routine invoke kernel software interrupt handler process try access page currently present page table vm area struct address space description contain field point table function implement key page- management functionality give virtual memory region request read write unavailable page eventually dispatch appro- priate handler function table vm area struct central memory management routine know detail manage possible type memory region virtual memory regions linux implement type virtual memory region property characterize virtual memory backing store region describe page region come memory region back file region back simple type virtual memory region region represent demand zero memory process try read page region simply give page memory fill zero region back file act viewport section file process try access page region page table fill address page kernel page cache correspond appropriate offset file page physical memory page cache process page table change file file system immediately visible process map file address space number process map region file end page physical memory purpose virtual memory region define reaction write mapping region process address space private share process write privately map region pager detect copy write necessary change local process contrast write share region result updating object map region change visible immediately process map object lifetime virtual address space kernel create new virtual address space situation process run new program exec system new process create fork system case easy new program execute process give new completely virtual address space routine load program populate address space virtual memory region second case create new process fork involve create complete copy exist process virtual address space kernel copy parent process vm area struct descriptor create new set page table child

vm area struct descriptor create new set page table child parent page table copy directly child reference count page cover incremente linux system fork parent child share physical page memory address space aspecial case occur copy operation reach virtual memory region map privately page parent process write region private subsequent change page parent child update page process address space page table entry region copy set read mark copy write long process modify page process share page physical memory process try modify copy write page reference count page check page share process copy page content brand new page physical memory use copy instead mechanism ensure private datum page share process possible copy swap paging important task virtual memory system relocate page memory physical memory disk memory need early unix system perform relocation swap content entire process modern version unix rely paging movement individual page virtual memory physical memory disk linux implement process swapping use new paging mechanism exclusively paging system divide section policy algorithm decide page write backing store write second paging mechanism carry transfer page datum physical memory need linux pageout policy use modify version standard clock second chance algorithm describe section 10.4.5.2 linux multiple pass clock page age adjust pass clock age precisely measure page youthfulness activity page see recently frequently access page attain high age value age infrequently access page drop zero pass age valuing allow pager select page page base frequently lfu paging mechanism support page dedicate swap device partition normal file swap file significantly slow extra overhead incur file system block allo- cat swap device accord bitmap block maintain physical memory time allocator use fit algo- rithm try write page continuous run secondary storage block improved performance allocator record fact page page storage feature page table modern proces- sors page table entry page present bit set modern proces- sors page table entry page present bit set allow rest page table entry fill index identify page kernel virtual memory linux reserve internal use constant architecture dependent region virtual address space process page table entry map kernel page mark protect page visible modifiable processor run user mode kernel virtual memory area contain region static area contain page table reference available physical page memory system simple translation physical virtual address occur kernel code run core kernel page allocate normal page allocator reside region remainder kernel reserved section address space reserve specific purpose page table entry address range modify kernel point area memory kernel provide pair facility allow kernel code use virtual memory vmalloc function allocate arbitrary number physical page memory physically contiguous single region virtually contiguous kernel memory vremap function map sequence virtual address point area memory device driver memory-execution loading user program linux kernel execution user program trigger exec system exec command kernel run new pro- gram current process completely overwrite current execution context initial context new program job system service verify call process permission right file execute matter check kernel invoke loader routine start run program loader necessarily load content program file physical memory set mapping program virtual memory single routine linux load new program instead linux maintain table possible loader function give function opportunity try load give file exec sys- tem initial reason loader table release 1.0 1.2 kernel standard format linux binary file change old linux kernel understand a.out format binary file relatively simple format common old unix system new linux system use modern elf format support current unix implementation elf number advantage a.out include flexibility extendability new section add elf binary example add extra debugging information cause loader rou- tine confused allow registration multiple loader routine linux easily support elf a.out binary format

multiple loader routine linux easily support elf a.out binary format single running section 20.6.3.1 section 20.6.3.2 concentrate exclusively loading running elf format binary procedure load a.out binary simple similar operation linux system mapping program memory linux binary loader load binary file physical mem- ory page binary file map region virtual memory program try access give page page fault result loading page physical memory demand paging responsibility kernel binary loader set initial memory mapping elf format binary file consist header follow page align section elf loader work read header map section file separate region virtual memory figure 20.6 show typical layout memory region set elf loader reserve region end address space sit kernel privileged region virtual memory inaccessible normal user- mode program rest virtual memory available application use kernel memory map function create region map portion file available application datum loader job set initial memory mapping allow execution program start region need initialize include stack program text datum region stack create user mode virtual memory grow downward lower number address include copy argu- ment environment variable give program exec system region create near end virtual memory section binary file contain program text read datum map memory write protect region writable initialize datum map uninitialized datum map private demand- kernel virtual memory memory invisible user mode code brk pointer memory layout elf program directly fix sized region variable sized region program expand need hold datum allocate run time process pointer brk point current extent datum region process extend contract brk region single system mapping set loader initialize process program counter register starting point record elf header process schedule static dynamic linking program load start run necessary content binary file load process virtual address space program need run function system library library function load simple case necessary library function embed directly program executable

load simple case necessary library function embed directly program executable binary file program statically link library statically link executable commence run soon load main disadvantage static linking program generate contain copy exactly common system library function efficient term physical memory disk space usage load system library memory dynamic linking allow happen linux implement dynamic linking user mode special linker library dynamically link program contain small statically link function call program start static function map link library memory run code function contain link library determine dynamic library require program name variable function need library read information contain section elf binary map library middle virtual memory resolve reference symbol contain library matter exactly memory share library map compile position independent code pic run address memory 20.7 file systems linux retain unix standard file system model unix file object store disk fetch network remote file server unix file capable handle input output stream datum device driver appear file interprocess- communication channel network connection look like file linux kernel handle type file hide implemen- tation detail single file type layer software virtual file system vfs cover virtual file system discuss standard linux file system ext3 linux system virtual file system linux vfs design object orient principle com- ponent set definition specify file system object allow look like layer software manipulate object vfs define main object type inode object represent individual file fil object represent open file superblock object represent entire file system dentry object represent individual directory entry object type vfs define set operation object type contain pointer function table function table list address actual function implement define operation object example abbreviated api file object operation include int open open file ssize t read read file ssize t write write file int mmap memory map file complete definition file object specify struct file operation

map file complete definition file object specify struct file operation locate file /usr include linux fs.h implementation file object specific file type require implement function specify definition file object vfs software layer perform operation file system object call appropriate function object function table have know advance exactly kind object deal vfs know care inode represent networked file disk file network socket directory file appropriate function file read operation place function table vfs software layer function care datum actually read inode file object mechanism access file inode object data structure contain pointer disk block contain actual file content file object represent point access datum open file thread access inode content obtain file object point inode file object keep track file process currently read write track sequential file o. remember permission example read write request file open track thread activity necessary perform adaptive read ahead fetch file datum memory thread request datum improve performance file object typically belong single process inode object file object instance open file single inode object file

long use process inode object cache vfs improve performance file near future cache file datum link list file inode object inode maintain standard information file owner size time recently modify directory file deal slightly differently file unix programming interface define number operation directory create delete rename file directory system call directory operation require user open file concern unlike case read write datum vfs define directory operation inode object file object superblock object represent connect set file form self contain file system operating system kernel maintain single superblock object disk device mount file system networked file system currently connect main responsibility superblock object provide access inode vfs identify inode unique file system inode number pair find inode correspond- e particular inode number ask superblock object return inode number finally dentry object represent directory entry include directory path file /usr actual file stdio.h example file /usr include stdio.h contain directory entry 1 2 usr 3 include 4 stdio.h value represent separate dentry object example dentry object consider situ- ation thread wish open file pathname /usr include stdio.h editor linux treat directory name file translate path require obtain inode root—/. operate system read file obtain inode file include continue thread obtain inode file stdio.h path translation time consume task linux maintain cache dentry object consult path translation obtain inode dentry cache considerably fast have read disk file linux ext3 file system standard disk file system linux call ext3 historical reason linux originally program minix compatible file sys- tem ease exchange datum minix development system file system severely restrict 14 character file limit maximum file system size 64 mb minix file system supersede new file system christen extended file system extfs later redesign improve performance scalability add miss feature lead second extended file system ext2 development add journal- e capability system rename extended file

ext2 development add journal- e capability system rename extended file system ext3 linux kernel developer augment ext3 modern file system feature extent new file system call fourth extend file system ext4 rest section discuss ext3 remain linux system deploy linux file system discussion apply equally linux ext3 common bsd fast file system ffs sec- tion c.7.7 use similar mechanism locate data block belong specific file store data block pointer indirect block file system level indirection ffs directory file store disk like normal file content interpret differently block directory file consist link list entry turn entry contain length entry file inode number inode entry refer main difference ext3 ffs lie disk allocation policy ffs disk allocate file block 8 kb block subdivide fragment 1 kb storage small file partially fill block end file contrast ext3 use fragment perform allocation small unit default block size ext3 vary function total size file system supported block size 1 2 4 8 kb maintain high performance operate system try perform o operation large chunk possible cluster physically adjacent o request clustering reduce request overhead incur device driver disk disk controller hardware block sized o request size small maintain good performance ext3 use allocation policy design place logically adjacent block file physically adjacent block disk submit o request disk block ext3 allocation policy work follow ffs ext3 file system partition multiple segment ext3 call block group ffs use similar concept cylinder group group correspond single cylinder physical disk note modern disk drive technol- ogy pack sector disk different density different cylinder size depend far disk head center disk fix sized cylinder group necessarily correspond disk geometry allocate file ext3 select block group file datum block attempt allocate file block group file inode allocate inode allocation select block group file parent directory reside nondirectory file directory file

block group file parent directory reside nondirectory file directory file keep disperse available block group policy design related information block group spread disk load disk block group reduce fragmentation area disk block group ext3 try allocation physically contiguous possible reduce fragmentation maintain bitmap free block block group allocate block new file start search free block beginning block group extend file continue search block recently allocate file search perform stage ext3 search entire free byte bitmap fail find look free bit search free byte aim allocate disk space chunk block possible free block identify search extend backward allocate block encounter free byte find bitmap backward extension prevent ext3 leave hole recently allocate block previous nonzero byte zero byte find block allocate find bit byte search ext3 extend allocation forward block preallocate extra block file preallocation help reduce fragmentation interleave write separate file reduce cpu cost disk allocation allocate multiple block simultaneously preallocate block return free space bitmap file close figure 20.7 illustrate allocation policy row represent sequence set unset bit allocation bitmap indicate free block disk case find free block sufficiently near start search allocate matter fragmented fragmentation partially compensate fact block close probably read disk seek furthermore allocate file well long run allocate isolate block separate file large free area scarce disk second case immediately find free block close search forward entire free byte bitmap allocate byte end create fragmented area free space allocation precede allocate allocation flush allocation precede allocate forward satisfy default allocation block allocate scatter free block allocate continuous free block block use ext3 block allocation policy linux system ext3 file system support popular feature call journaling modification file system write sequentially journal set operation perform specific task transaction transaction write journal consider commit journal entry relate transaction replay actual file- system structure change pointer update indicate

transaction replay actual file- system structure change pointer update indicate action complete incomplete entire commit transaction complete remove journal jour- nal actually circular buffer separate section file system separate disk spindle efficient complex separate read write head decrease head contention seek time system crash transaction remain journal transaction complete file system commit operate system complete system recover transaction execute pointer work complete file system structure remain consistent problem occur transaction abort commit system crash change transaction apply file system undo preserve consistency file system recovery need crash eliminate problem consistency checking journale file system perform operation fast nonjour- nale system update proceed fast apply memory journal directly disk datum structure rea- son improvement find performance advantage sequential o random o.

costly synchronous random write file system turn costly synchronous sequential write file sys- tem journal change turn replay asynchronously random write appropriate structure overall result significant gain performance file system metadata orient operation file creation deletion performance improvement ext3 configure journal metadata file datum linux proc file system flexibility linux vf enable implement file system store datum persistently provide interface functionality linux /proc file system example file system content actually store compute demand accord user file o request proc file system unique linux unix v8 introduce /proc file system use adopt expand operate system efficient interface kernel process space help debugging subdirectory file system correspond directory disk active process current system listing file system reveal directory process directory ascii decimal representation process unique process linux implement /proc file system extend greatly add number extra directory text file file system root direc- tory new entry correspond statistic kernel associate load driver /proc file system provide way pro- grams access information plain text file standard unix user environment provide powerful tool process file example past traditional unix ps command list state run process implement privileged process read process state directly kernel virtual memory linux command implement entirely unprivileged program simply parse format information /proc /proc file system implement thing directory structure file content unix file system define set file directory inode identify inode number /proc file system define unique persistent inode number directory associate file mapping exist file system use inode number identify operation require user try read particular file inode perform lookup particular directory inode datum read file /proc file system collect appropriate information format textual form place request process read buffer mapping inode number information type split inode number field linux pid 16 bit size inode number 32 bit 16 bit inode number interpret pid remain bit define type information

bit inode number interpret pid remain bit define type information request pid zero valid zero pid field inode number take mean inode contain global process specific information separate global file exist /proc report information kernel version free memory performance statistic driver currently inode number range reserve kernel allocate new /proc inode mapping dynamically maintain bitmap allocate inode number maintain tree datum structure register global /proc file system entry entry contain file inode number file access permission special function gen- erate file content driver register deregister entry tree time special section tree appear /proc sys directory reserve kernel variable file tree manage set common handler allow reading writing vari- able system administrator tune value kernel parameter simply write new desire value ascii decimal appropriate file allow efficient access variable application /proc sys subtree available special system sysctl read write variable binary text overhead file system sysctl extra facility simply read /proc dynamic entry tree identify variable application linux system device driver block structure 20.8 input output user o system linux look like unix system extent possible device driver appear normal file user open access channel device way open file device appear object file system system admin- istrator create special file file system contain reference specific device driver user open file able read write device reference normal file protection system determine access file administrator set access permission device linux split device class block device character device network device figure 20.8 illustrate overall structure device- block device include device allow random access completely independent fix sized block datum include hard disk floppy disk cd rom blu ray disc flash memory block device typically store file system direct access block device allow program create repair file system device contain application access block device directly wish example database application prefer perform fine tune layout datum

wish example database application prefer perform fine tune layout datum disk general purpose file system character device include device mouse keyboard fundamental difference block character device random access block device access randomly character device access serially example seek certain position file support dvd make sense point device mouse network device deal differently block character device user directly transfer datum network device instead communicate indirectly open connection kernel net- work subsystem discuss interface network device separately block device provide main interface disk device system perfor- mance particularly important disk block device system input output provide functionality ensure disk access fast possible functionality achieve scheduling o operation context block device block represent unit kernel perform o. block read memory store buffer request manager layer software manage reading writing buffer content block device driver separate list request keep block device driver tradition- ally request schedule accord unidirectional elevator c scan algorithm exploit order request insert remove list request list maintain sorted order increase start sector number request accept processing block device driver remove list remove o complete point driver continue request list new request insert list active request new o request request manager attempt merge request list linux kernel version 2.6 introduce new o scheduling algorithm simple elevator algorithm remain available default o sched- uler completely fair queueing cfq scheduler cfq o scheduler fundamentally different elevator base algorithm instead sort request list cfq maintain set list default process request originate process process list example process issue o request cfq maintain sep- arate list request process list maintain accord c scan algorithm cfq service list differently traditional c scan algo- rithm indifferent specific process cfq service process list round- robin pull configurable number request default list move method result fairness process level process receive equal fraction disk bandwidth result beneficial interactive workload o latency equal fraction disk bandwidth result beneficial interactive workload o latency important practice cfq perform workload character device driver device driver offer random access fix block datum character device driver register linux kernel register set function implement file o operation driver handle kernel perform preprocessing file read write request character device simply pass request device question let device deal main exception rule special subset character device driver implement terminal device kernel maintain standard interface driver mean set tty struct structure structure provide buffering flow control data stream terminal device feed datum line discipline line discipline interpreter information terminal device common line discipline tty discipline glue terminal data stream standard input output stream user run process allow process communicate directly linux system user terminal job complicate fact process run simultaneously tty line discipline responsible attach detach terminal input output process connect process suspend awaken line discipline implement o user process ppp slip networking protocol way encode networking connection terminal device serial line protocol implement linux driver end appear terminal system line discipline end appear network system network device driver line discipline enable terminal device datum appear terminal route directly appropriate network device driver 20.9 interprocess communication linux provide rich environment process communicate communication matter let process know event occur involve transfer datum process synchronization signal standard linux mechanism inform process event occur signal signal send process process restriction signal send process own user limited number signal available carry information fact signal occur available process signal generate process kernel generate signal internally example send signal server process datum arrive network channel parent process child terminate wait process timer expire internally linux kernel use signal communicate process run kernel mode kernel mode process expect event occur use signal receive notification event communication incoming asynchronous event kernel take place use scheduling state wait queue structure

event kernel take place use scheduling state wait queue structure mechanism allow kernel mode process inform relevant event allow event generate device driver network system process want wait event complete place wait queue associate event tell scheduler long eligible execution event complete process wait queue awaken procedure allow multiple process wait single event example process try read file disk awaken datum read memory successfully signal main mechanism commu- nicate asynchronous event process linux implement semaphore mechanism system v unix process wait semaphore easily wait signal semaphore advantage large number semaphore share multiple independent process operation multiple semaphore perform atomically inter- nally standard linux wait queue mechanism synchronize process communicate semaphore passing datum processes linux offer mechanism pass datum process stan- dard unix pipe mechanism allow child process inherit communication channel parent datum write end pipe read linux pipe appear type inode virtual file system software pipe pair wait queue synchronize reader writer unix define set networking facility send stream datum local remote process networking cover process communication method shared memory offer extremely fast way communicate large small amount datum datum write process share memory region read immediately process map region address space main disadvantage shared memory offer synchronization process ask operate system piece shared memory write suspend execution write occur shared memory particularly powerful conjunction interprocess communication mechanism provide miss synchronization share memory region linux persistent object create delete process object treat small independent address space linux paging algorithm elect page share memory page disk page process datum page share memory object act backing store share memory region file act backing store memory map memory region file map virtual address space region page fault occur cause appropriate page file map virtual memory similarly share memory mapping direct page fault map page persistent share memory object file shared- memory object remember

page persistent share memory object file shared- memory object remember content process currently map virtual memory 20.10 network structure networking key area functionality linux

linux support standard internet protocol unix unix communi- cation implement number protocol native non unix operate system particular linux originally implement pri- marily pc large workstation server class system support protocol typically pc network appletalk ipx internally networking linux kernel implement layer linux system 1 socket interface 2 protocol driver 3 network device driver user application perform networking request socket interface interface design look like 4.3 bsd socket layer program design use berkeley socket run linux source code change interface describe section c.9.1 bsd socket interface sufficiently general represent network address wide range networking protocol single interface linux access protocol implement standard bsd system protocol support system layer software protocol stack similar orga- nization bsd framework network datum arrive layer application socket network device driver datum expect tag identifier specify network protocol contain protocol communicate desire example internet protocol set separate protocol manage routing error reporting reliable retransmission lose datum protocol layer rewrite packet create new packet split reassemble packet fragment simply discard incoming datum ultimately protocol layer finish process set packet pass upward socket interface datum destine local connection downward device driver datum need transmit remotely protocol layer decide socket device send packet communication layer networking stack per- form pass single skbuff socket buffer structure structure contain set pointer single continuous area memory represent buffer inside network packet construct valid datum skbuff need start beginning skbuff buffer need run end networking code add datum trim datum end packet long result fit skbuff capacity especially important modern microproces- sor improvement cpu speed far outstrip performance main memory skbuff architecture allow flexibility manipulate packet header checksum avoid unnecessary data copying important set protocol linux network system tcp

unnecessary data copying important set protocol linux network system tcp ip protocol suite suite comprise number separate protocol ip protocol implement routing different host network routing protocol udp tcp icmp protocol udp protocol carry arbitrary individual datagram host tcp protocol implement reliable connection host guaranteed order delivery packet automatic retransmission lose datum icmp protocol carry error status message host packet skbuff arrive network stack protocol software expect tag internal identifier indicate protocol packet relevant different networking device driver encode protocol type different way protocol incoming datum identify device driver device driver use hash table know networking protocol identifier look appropriate protocol pass packet protocol new protocol add hash table kernel loadable module incoming ip packet deliver ip driver job layer perform routing decide packet send ip driver forward packet appropriate internal protocol driver deliver locally inject select network device driver queue forward host perform routing decision table persistent forwarding information base fib cache recent routing decision fib hold routing configuration information specify route base specific destination address wildcard represent multiple destination fib organize set hash table index destination address table represent specific route search successful lookup table add route caching table cache route specific destination wildcard store cache lookup quickly entry route cache expire fix period hit stage ip software pass packet separate section code firewal management selective filtering packet accord arbitrary criterion usually security purpose firewall manager maintain number separate firewal chain allow skbuff match chain chain reserve separate purpose forward packet packet input host datum generate host chain hold ordered list rule rule specify number possible firewall decision function plus arbitrary datum matching purpose function perform ip driver disassembly reassembly large packet outgoing packet large queue device simply split small fragment queue driver receive host fragment reassemble ip driver maintain ipfrag object fragment await reassembly ipq datagram

ip driver maintain ipfrag object fragment await reassembly ipq datagram assemble incoming fragment match know ipq match find fragment add oth- erwise new ipq create final fragment arrive ipq completely new skbuff construct hold new packet packet pass ip driver packet identify ip destine host pass protocol driver udp tcp protocol share means associate packet source destination socket connect pair socket uniquely identify source destination address source destination port number socket list link hash table key address port value socket lookup incoming packet tcp protocol deal unreliable connection maintain order list unacknowledged outgoing packet retransmit timeout incoming order packet present socket miss datum arrive linux system linux security model closely relate typical unix security mechanism security concern classify group 1 authentication make sure access system prove entry right 2 access control provide mechanism check user right access certain object prevent access object authentication unix typically perform use publicly readable password file auser password combine random salt value result encode way transformation function store password file use way function mean original password deduce password file trial error user present password system password recombine salt value store password file pass way transformation result match content password file password accept historically unix implementation mechanism drawback password limit character number possible salt value low attacker easily combine dictionary commonly password possible salt value good chance match password password file gain unauthorized access account compromise result extension password mechanism introduce encrypt password secret file publicly readable allow long password use secure method encode password authentication mechanism introduce limit period user permit connect system mechanism exist distribute authentication information relate system anew security mechanism develop unix vendor address authentication problem pluggable authentication module pam sys- tem base shared library system component need authenticate user implementation system available linux pam allow authentication module load demand specify system

available linux pam allow authentication module load demand specify system wide configuration file new authentication mecha- nism add later date add configuration file system component immediately able advantage pam mod- ule specify authentication method account restriction session setup function password change function user change password necessary authentication mechanism update access control unix system include linux perform use unique numeric identifier auser identifier uid identify single user single set access right group identifier gid extra identifier identify right belong user access control apply object system file available system protect standard access control mecha- nism addition share object share memory section

semaphore employ access system object unix system user group access control single uid single gid associate user process single uid gid process uid match uid object process user right owner right object uid match gid process match object gid group right confer process world right linux perform access control assign object protection mask specify access mode read write execute grant process owner group world access owner object read write execute access file user certain group give read access deny write access everybody give access exception privileged root uid process special uid grant automatic access object system bypass normal access check process grant permission perform privi- leged operation read physical memory open reserve network socket mechanism allow kernel prevent normal user access resource kernel key internal resource implicitly own root uid linux implement standard unix setuid mechanism describe sec- tion c.3.2 mechanism allow program run privilege different user run program example lpr program submit job print queue access system print queue user run program unix implementation setuid distinguish process real effective uid real uid user run program effective uid file linux mechanism augment way linux implement posix specification save user id mechanism allow process drop reacquire effective uid repeatedly security reason program want perform operation safe mode waive privilege grant setuid status wish perform select operation privilege standard unix implementation achieve capacity swap real effective uid previous effective uid remember program real uid correspond uid user run program save uid allow process set effective uid real uid return linux system previous value effective uid have modify real uid time second enhancement provide linux addition process characteristic grant subset right effective uid fsuid fsgid process property access right grant file appropriate property set time effective uid gid set fsuid fsgid set independently effective id

allow process access file behalf user take identity user way specifically server process use mechanism serve file certain user vulnerable kill suspend user finally linux provide mechanism flexible passing right program mechanism common modern version unix local network socket set process system process send process file descriptor open file process receive duplicate file descriptor file mechanism allow client pass access single file selectively server process grant process privilege example long necessary print server able read file user submit new print job print client simply pass server file descriptor file print deny server access user file linux modern free operate system base unix standard design run efficiently reliably common pc hardware run variety platform mobile phone provide programming interface user interface compatible standard unix system run large number unix application include increase number commercially support application linux evolve vacuum complete linux system include component develop independently linux core linux operate system kernel entirely original allow existing free unix software run result entire unix compatible operate system free proprietary code linux kernel implement traditional monolithic kernel performance reason modular design allow driver dynamically load unload run time linux multiuser system provide protection process run multiple process accord time sharing scheduler newly create process share selective part execution environment parent process allow multithreaded programming interprocess communication support system v mechanism message queue semaphore share memory bsd socket interface multiple networking protocol access simultaneously socket interface memory management system use page sharing copy write minimize duplication datum share different process page load demand reference page back store accord lfu algorithm physical memory need reclaim user file system appear hierarchical directory tree obey unix semantic internally linux use abstraction layer man- age multiple file system device orient networked virtual file system support device orient file system access disk storage page cache unify virtual memory system

system access disk storage page cache unify virtual memory system dynamically loadable kernel module flexibility driver add system disadvantage circumstance kernel compile single binary file well split module explain multithreading commonly programming technique describe different way implement thread compare method linux clone mechanism alternative mechanism well bad clone linux kernel allow paging kernel memory effect restriction kernel design advantage disadvantage design decision discuss advantage dynamic share linkage library com- pare static linkage describe case static linkage compare use networking socket use share memory mechanism communicate datum process single computer advantage method time unix system disk layout optimization base rotation position disk datum modern implementation includ- e linux simply optimize sequential datum access hardware characteristic sequential access advantage rotational optimization long useful linux system product internet result avail- able documentation linux available form internet following key site reference useful information available linux system linux cross reference page lxr http://lxr.linux.no

maintain cur- rent listing linux kernel browsable web fully cross- kernel hackers guide provide helpful overview linux kernel component internal locate http://tldp.org/ldp/tlk/tlk.html linux weekly news lwn http://lwn.net provide weekly linux- relate news include research subsection linux kernel mailing list devote linux available important maintain mailing list manager reach e mail address majordomo@vger.rutgers.edu send e mail address single line help mail body information access list server subscribe list finally linux system obtain internet com- plete linux distribution available home site compa- ny concern linux community maintain archive current system component place internet important addition investigate internet resource read internal linux kernel mauerer 2008 love 2010 /proc file system introduce http://lucasvr.gobolinux.org/etc/killian84-procfs-usenix.pdf expand http://https://www.usenix.org sites default/files usenix winter91 faulkner.pdf r. love linux kernel development edition developer w. mauerer professional linux kernel architecture john wiley sons 2008 chapter 20 exercise advantage disadvantage write operate system high level language c circumstance system sequence fork()exec appropriate vfork preferable socket type implement intercomputer file- transfer program type program peri- odically test computer network explain answer linux run variety hardware platform step linux developer ensure system portable different pro- cessor memory management architecture minimize architecture specific kernel code advantage disadvantage make symbol define inside kernel accessible loadable kernel module primary goal conflict resolution mechanism linux kernel load kernel module discuss clone operation support linux sup- port process thread classify linux thread user level thread kernel- level thread support answer appropriate argument extra cost incur creation scheduling pro- cess compare cost clone thread linux

completely fair scheduler cfs provide improve fairness traditional unix process scheduler fairness configurable variable completely fair sched- uler cfs pro con set small large value linux scheduler implement soft real time scheduling fea- ture necessary certain real time programming task miss add kernel cost downside feature circumstance user process request operation result allocation demand zero memory region scenario cause page memory map user program address space copy write attribute linux share library perform operation central oper- ating system advantage keep functionality kernel drawback explain answer benefit journaling file system linux ext3 cost ext3 provide option journal directory structure linux operate system include file correspond different file system include linux /proc file system need support different file system type affect structure linux kernel way linux setuid feature differ setuid linux source code freely widely available internet cd rom vendor implication avail- ability security linux system c h p t e r update alex ionescu microsoft windows 10 operating system preemptive multitaske client operating system microprocessor implement intel ia-32 amd64 arm arm64 instruction set architecture isas microsoft corre- sponde server operate system windows server 2016 base code windows 10 support 64 bit amd64 isas windows 10 late series microsoft operating system base nt code replace early system base windows 95/98 chapter discuss key goal windows 10 layered architecture system easy use file system networking feature programming interface explore principle underlie windows 10 design specific component system provide detailed discussion windows 10 file system illustrate networking protocol support windows 10 describe interface available windows 10 system application describe important algorithm implement windows 10 system write assembly language single processor intel develop new technology nt portable operate system support os/2 posix application programming interface apis october 1988 dave cutler architect dec vax vms operate system hire give charter build microsoft new operating system originally

system hire give charter build microsoft new operating system originally team plan use os/2 api nt native environ- ment development nt change use new 32 bit windows api call win32 base popular 16 bit api windows 3.0 version nt windows nt 3.1 windows nt 3.1 advanced server time 16 bit windows version 3.1 windows nt ver- sion 4.0 adopt windows 95 user interface incorporate internet web server web browser software addition user interface routine graphic code move kernel improve performance effect decrease system reliability significant loss security previous version nt port microprocessor architecture include brief 64 bit port alpha axp 64 windows 2000 version release february 2000 support ia-32 compatible pro- cessor marketplace factor windows 2000 incorporate significant change add active directory x.500 base directory service well networking laptop support support plug play device dis- tribute file system support processor memory windows xp vista 7 october 2001 windows xp release update windows 2000 desktop operate system replacement windows 95/98 april 2003 server edition windows xp call windows server 2003 available windows xp update graphical user interface gui visual design take advantage recent hardware advance new ease use feature numerous feature add automatically repair problem application operate system change windows xp provide well networking device experience include zero configuration wireless instant messaging stream medium digital photography video windows server 2003 provide dramatic performance improvement large multiprocessor system well reliability security early windows operate system long await update windows xp call windows vista release january 2007 receive windows vista include improvement later continue windows 7 improvement overshadow windows vista perceive sluggish- ness compatibility problem microsoft respond criticism win- dows vista improve engineering process work closely maker windows hardware application result windows 7 release october 2009 corresponding server edition call windows server 2008 r2 significant engineering change increase use event tracing counter profiling

r2 significant engineering change increase use event tracing counter profiling analyze system behavior tracing run constantly system watch hundred scenario execute scenario include process startup exit file copy web page load example scenario fail succeed perform trace analyze determine cause year later october 2012 amid industry wide pivot mobile computing world app microsoft release windows 8 represent significant change operate system windows xp windows 8 include new user interface name metro new programming model api name winrt include new way manage application run new sandbox mechanism package system exclusively support new windows store com- petitor apple app store android store additionally windows 8 include plethora security boot performance improvement time support subsystem concept describe later chapter remove support new mobile world windows 8 port 32 bit arm isa time include multiple change power man- agement hardware extensibility feature kernel discuss later chapter microsoft market version port version call windows rt run windows store package application microsoft brand classic application notepad internet explorer importantly office version call windows phone run windows store package application time microsoft release brand mobile hard- ware surface brand include surface rt tablet device exclusively run windows rt operate system bit later microsoft buy nokia begin release microsoft brand phone run unfortunately windows 8 market failure reason hand metro focus tablet orient interface force user accus- tome old windows operate system completely change way work desktop computer windows 8 example replace start menu touchscreen feature replace shortcut animate tile offer little keyboard input support hand dearth application windows store way obtain app microsoft phone tablet lead market failure device cause company eventually phase surface rt device write nokia purchase microsoft quickly seek address issue release windows 8.1 october 2013 release address usabil- ity flaw

windows 8 nonmobile device bring usability traditional keyboard mouse provide way avoid tile base metro interface continue improve security performance reliability change introduce windows 8 release well receive continue lack application windows store problem operate system mobile market penetration desktop server application programmer feel abandon lack improvement area release windows 10 july 2015 server companion windows server 2016 october 2016 microsoft shift windows- service waas model include periodic functionality improve- ment

windows 10 receive monthly incremental improvement call fea- ture rollup month feature release call update addi- tionally upcoming release available public windows insider program wip release version weekly basis like cloud service website facebook google new operate system use live telemetry send debug information microsoft tracing dynamically enable disable certain feature b testing compare version execute compare similar version b try new feature watch compatibility issue aggressively add remove support modern legacy hardware dynamic configuration testing feature release service implementation windows 10 reintroduce start menu restore keyboard support deemphasize screen application live tile user perspec- tive change bring ease use user expect windows base desktop operate system additionally metro rename modern redesign windows store package applica- tion run regular desktop legacy application finally new mechanism call windows desktop bridge pos- sible place win32 application windows store mitigate lack application write specifically new system microsoft add support c++11 c++14 c++17 visual studio product new api add traditional win32 programming api related change windows 10 release unified windows platform uwp architecture allow application write way execute windows desktop windows iot xbox windows phone windows 10 mixed reality previously know windows windows 10 replace concept multiple subsystem remove windows 8 mention early new mechanism call pico providers mechanism allow unmodified binary belong different operate system run natively windows 10 anniver- sary update release august 2016 functionality provide windows subsystem linux run linux elf binary entirely unmodified ubuntu user space environment response increase competitive pressure mobile cloud- compute world microsoft power performance scalability improvement windows 10 enable run large number device fact version call windows 10 iot edition specifically design environment raspberry pi support cloud computing technology containerization build docker win- dow windows 10 microsoft hyper v virtualization technology build provide additional security native support run virtual machine special version windows server call

native support run virtual machine special version windows server call windows server nano release extremely low overhead server operating system suit containerized application cloud computing usage windows 10 multiuser operating system support simultaneous access distribute service multiple instance gui windows terminal services server edition windows 10 support simultaneous terminal server session windows desktop system desktop edition terminal server multiplex keyboard mouse mon- itor virtual terminal session log user feature call fast user switching allow user preempt console pc have log log let return briefly development windows gui note early gui implementation move kernel mode windows nt 4.0 improve performance performance gain creation new user mode component windows vista call desktop window manager dwm dwm provide windows interface look feel windows directx graphic software directx continue run kernel code win32k implement windows windowing graphic model user gdi windows 7 substantial change dwm sig- nificantly reduce memory footprint improve performance windows 10 improvement especially area perfor- mance security furthermore windows directx 11 12 include gpgpu mechanism general purpose computing gpu hardware direct- compute part windows update advantage high performance graphic model new rendering layer call coreui legacy application advantage directx base rendering creation final screen content windows xp version windows ship 64 bit version ia64 2003 amd64 2005 internally native nt file system ntfs win32 api 64 bit integer appropriate major extension 64 bit windows xp mean support large virtual address addition 64 bit edition windows support large physical memory late windows server 2016 release support 24 tb ram time windows 7 ship amd64 isa available cpu intel amd addition time physical memory client system frequently exceed 4 gb limit ia-32 result 64 bit version windows 10 exclusively instal client system apart iot mobile system amd64 architecture support high fidelity ia-32 compatibility level individual process 32- 64 bit application freely mix single

level individual process 32- 64 bit application freely mix single system interestingly similar pattern emerge mobile system apple ios mobile operate system support arm64 architecture 64 bit isa extension arm call aarch64 afuture windows 10 release officially ship arm64 port design new class hardware compatibility ia-32 architecture application achieve emulation dynamic jit rest description windows 10 distinguish client edition corresponding server edition base core component run binary file kernel driver similarly microsoft ship variety different edition release address different market price point difference edition reflect core system chapter focus primarily core component windows 10 21.2 design principles microsoft design goal windows include security reliability compati- bility high performance extensibility portability international support additional goal energy efficiency dynamic device support recently add list discuss goal achieve windows 10 windows vista later security goal require adherence design standard enable windows nt 4.0 receive c2 secu- rity classification u.s. government c2 classification signify moderate level protection defective software malicious attack classification define department defense trusted com- puter system evaluation criteria know orange book extensive code review testing combine sophisticated automatic analysis tool identify investigate potential defect represent security vulnerability additionally bug bounty participation program allow exter- nal researcher security professional identify submit previously unknown security issue windows exchange receive monetary payment credit monthly security rollup release microsoft windows 10 secure possible windows traditionally base security discretionary access control sys- tem object include file registry key kernel synchronization object protect access control list acls section 13.4.2 acl vul- nerable user programmer error com- mon attack consumer system user trick run code browse web windows vista introduce mechanism call integrity level act rudimentary capability system con- trolling access object process mark have low medium high system integrity integrity level determine right object process example windows allow process modify object high integrity level base mandatory policy

allow process modify object high integrity level base mandatory policy matter setting acl additionally process read memory high integrity process matter acl windows 10 strengthen security model introduce combination attribute base access control abac claim base access control cabc feature implement dynamic access control dac server edition support capability base system windows store application modern package application attribute claim system administrator need rely user group user belong mean security system use filter access object file property user seniority organization salary consider property encode attribute pair conditional access control entry acl seniority > = 10 year windows use encryption common protocol communicate securely website encryption protect user file store secondary storage windows 7 later version allow user easily encrypt entire volume removable storage device usb flash drive feature call bitlocker computer encrypt volume steal thief need sophisticated technology electron microscope gain access computer file impossible user configure external usb base token usb token steal type security feature focus user datum security vulnerable highly privileged program parse arbitrary content trick programming error execute malicious code windows include security measure refer exploit mitigation measure include wide scope mitigation address space layout randomization aslr data execution prevention dep control flow guard cfg arbitrary code guard acg narrow scope target mitigation specific exploitation technique outside scope chapter 2001 chip intel amd allow memory page mark contain executable instruction code win- dow dep feature mark stack memory heap data- allocation execute code prevent attack program bug allow buffer overflow trick execute content buffer additionally start windows 8.1 kernel datum memory allocation mark similarly dep prevent attacker control datum execute code malicious developer move code reuse attack exist- e executable code inside program reuse unexpected way certain part code execute flow redirect instruction stream aslr thwart form attack randomize location executable datum region memory make hard code

attack randomize location executable datum region memory make hard code reuse attack know existing code locate safeguard make likely system attack remote attacker fail crash mitigation perfect aslr exception example ineffective local attack application trick load content secondary storage example call information leak attack program trick reveal address space address problem windows 8.1 introduce technology call cfg improve windows 10 cfg work compiler linker loader memory manager validate destination address indirect branch jump list valid function prologue program trick redirect control flow instruction crash attacker bring executable datum attack reuse existing code attempt cause program allocate executable writeable code fill attacker alternatively attacker modify exist writeable datum mark executable datum windows 10 acg mitigation prohibit operation executable code load modify datum load mark executable windows 10 thirty security mitigation addition describe set security feature traditional attack difficult explain crimeware application adware credit card fraudware ransomware prevalent type attack rely user willingly manually cause harm computer double click application warn inputte credit card number fake banking page operate system design militate gullibility curios- ity human being recently microsoft start work directly chip manufacturer intel build security mitigation directly isa mitigation example control flo cet hardware implementation cfg protect return orient programming rop attack hardware shadow stack shadow stack contain set return address store routine call address check mismatch return exe- cut mismatch mean stack compromise action important aspect security integrity windows offer digital signature facility code integrity feature windows use digital signature sign operating system binary verify file produce microsoft know company non ia-32 version windows code integrity module activate boot ensure load module kernel valid signature assure tamper additionally arm version windows 8 extend code integrity module user mode code integrity check validate user program sign microsoft deliver windows store special version windows 10 windows 10 s mean education market

special version windows 10 windows 10 s mean education market provide similar signing check ia-32 amd64 system digital signature code integrity guard allow application defend load- e executable code secondary storage appropriately sign example attacker replace party binary digital signature fail code integrity guard load binary process address space finally enterprise version windows 10 possible opt new security feature call device guard mechanism allow organiza- tion customize digital signing requirement computer system blacklist whitelist individual signing certificate binary hash example organization choose allow user mode program sign microsoft google adobe launch enterprise windows mature greatly operate system year lead windows 2000 time reliability increase factor maturity source code extensive stress testing system improved cpu architecture automatic detection error driver microsoft party windows subsequently extend tool achieve reliability include automatic analysis source code error test detect validation failure application version driver verifier apply dynamic checking common user mode programming error improvement reliability result move code kernel user mode service windows provide extensive support write driver user mode system facility kernel user mode include renderer party font software stack audio significant improvement windows experience come add memory diagnostic option boot time addition especially valuable consumer pc error correct mem- ory bad ram lack error correction detection change datum store change undetected hardware result frustratingly erratic behavior system availability memory diagnostic warn user ram problem windows 10 take introduce run- time memory diagnostic machine encounter kernel mode crash time row crash pinpoint specific cause component kernel use idle period memory content flush system cache write repeat memory testing pattern memory preemptively discover ram damage user inform issue need reboot memory diagnostic tool boot windows 7 introduce fault tolerant memory heap heap learn application crash automatically adjust memory operation carry application crash make application reliable contain common bug memory free access past end allocation

reliable contain common bug memory free access past end allocation bug exploit attacker windows 7 include mitigation developer block feature immediately crash application heap corruption practical representation dichotomy exist need security need user experience achieve high reliability windows particularly challenging billion system run windows reliability problem affect small percentage system impact tremendous number user complexity windows ecosystem add challenge million instance application driver software constantly download run windows system course constant stream malware attack windows hard attack directly exploit increasingly target popular application cope challenge microsoft increasingly rely com- munication customer machine collect datum ecosystem machine sample perform software run problem encounter automatically send datum microsoft software driver kernel crash hang feature measure indicate legacy behavior method long recommend use microsoft disable alert send attempt use result microsoft build improve picture happen windows ecosystem allow continuous improvement software update provide datum guide future release windows application compatibility mention windows xp update windows 2000 replacement windows 95/98 windows 2000 focus primarily compat- ibility business application requirement windows xp include high compatibility consumer application run win- dows 95/98 application compatibility difficult achieve rea- son example application check specific version windows depend extent quirk implementation api latent application bug mask previous system application compile different instruction set different expectation run today multi gigahertz multicore system windows 10 continue focus compatibility issue implement- e strategy run application despite incompatibility like windows xp windows 10 compatibility layer call shim engine sit application win32 api engine windows 10 look bug bug compatible previous ver- sion windows windows 10 ship shim database 6,500 entry describe particular quirk tweak old application furthermore application compatibility toolkit user administrator build shim database windows 10

switchbranch mechanism allow developer choose windows ver- sion like win32 api emulate include quirk and/or bug previous api task manager operating system context col- umn show switchbranch operating system version application windows 10 like early nt release maintain support run 16 bit application thunke conversion layer call windows- windows-32 wow32)—that translate 16 bit api call equivalent 32- bit call similarly 64 bit version windows 10

provide thunking layer wow64 translate 32 bit api call native 64 bit call finally arm64 version windows 10 provide dynamic jit recompiler translate ia-32 code call wowa64 original windows subsystem model allow multiple operating system personality support long application rebuild portable executable pe application microsoft compiler visual stu- dio source code available note early api design windows win32api early edition windows support posix subsystem posix standard specification unix allow unix- compatible software recompile run modification posix compatible operating system unfortunately linux mature drift far far away posix compatibility mod- ern linux application rely linux specific system call improve- ment glibc standardized additionally impractical ask user enterprise recompile visual studio single linux application like use compiler difference ble subsystem model exist architectural level subsystem windows go forward win32 subsys- tem compatibility operate system achieve new model use pico providers instead significantly powerful model extend kernel ability forward proxy system exception fault thread creation termination process creation internal operation secondary external driver pico provider secondary driver owner operation windows 10 scheduler memory manager similar microkernel implement abi system interface executable file format parser page fault handling caching o model security model windows 10 include pico provider call lxcore multi- megabyte reimplementation linux kernel note linux share code linux driver windows subsystem linux feature load unmodified linux elf binary need source code recompilation pe binary windows 10 user run unmodified ubuntu user mode file system recently opensuse centos service apt package management command run package normal note ker- nel reimplementation complete system call miss access device linux kernel driver load notably networking fully support serial device gui frame buffer access possible final compatibility measure windows 8.1 later version include hyper v client feature allow application bug- bug compatibility windows xp linux dos run operate system inside virtual machine windows design provide high performance

operate system inside virtual machine windows design provide high performance desktop system largely constrain o performance server system cpu bottleneck large multithreaded multiprocessor envi- ronment locking performance cache line management key scalability satisfy performance requirement nt variety tech- nique asynchronous o optimize protocol network kernel- base graphic rendering sophisticated caching file system datum memory management synchronization algorithm design awareness performance consideration relate cache line multi- windows nt design symmetrical multiprocessing smp multiprocessor computer thread run time kernel cpu windows nt use priority base preemptive scheduling thread execute dispatcher interrupt level thread process run windows preempt higher-priority thread system respond quickly chapter 5 windows xp improve performance reduce code path length critical function implement scalable locking protocol queue spinlock pushlock pushlock like optimize spin- lock read write lock feature new locking protocol help reduce system bus cycle include lock free list queue atomic read modify write operation like interlock increment advanced syn- chronization technique change need windows xp add support simultaneous multithreading smt massively parallel pipelining technology intel commercialize mar- keting hyper threading new technology average home machine appear processor year later introduc- tion

multicore system multiprocessor system norm windows server 2003 target large multiprocessor server release well algorithm make shift per- processor data structure lock cache page coloring support numa machine page coloring performance optimization ensure access contiguous page virtual memory optimize use processor cache windows xp 64 bit edition base windows server 2003 kernel early 64 bit adopter advantage time windows 7 develop major change come computing number cpu physical memory available large multiprocessor increase substantially lot effort improve operate system scalability implementation multiprocessing support windows nt bit- mask represent collection processor identify example set processor particular thread schedule bitmask define fit single word memory limit number processor support system 64 64 bit system 32 32 bit system windows 7 add concept processor group represent collection 64 processor multiple processor group create accommodate total 64 processor note windows call schedulable portion processor execution unit logical processor distinct physical processor core refer processor cpu chapter mean logical processor windows point view windows 7 support processor group total 256 logical processor windows 10 support 20 group total 640 logical processor group additional cpu create great deal contention lock schedule cpu memory windows 7 break lock apart example windows 7 single lock windows scheduler synchronize access queue contain thread wait event windows 7 object lock allow queue access concurrently similarly global object manager lock cache manager vacb lock memory manager pfn lock synchronize access large global datum structure decompose lock small data structure execution path scheduler rewrite lock free change result improved scalability performance windows 7 system 256 logical cpu change increase importance support par- allel computing year computer industry dominate moore law section 1.1.3 lead high density transistor manifest fast clock rate cpu moore law contin- ue hold true limit reach prevent cpu clock rate increase instead transistor build cpu chip new programming model

rate increase instead transistor build cpu chip new programming model achieve paral- lel execution microsoft concurrency runtime concrt par- allel processing library ppl intel threading building blocks tbb express parallelism c++ program additionally vendor neutral standard call openmp support compiler moore law govern computing year amdahl law govern parallel computing section 4.2 rule future finally power consideration complicate design decision high performance computing especially mobile system battery life trump performance need cloud server environment cost electricity outweigh need fast possi- ble computational result accordingly windows 10 support feature sacrifice raw performance well power efficiency example include core parking put idle system sleep state heterogeneous multi processing hmp allocate task efficiently support task base parallelism amd64 port windows 7 later version provide new form user mode scheduling ums ums allow program decompose task task schedule available cpu scheduler operate user mode advent multiple cpu small computer shift take place parallel computing graphic process unit gpu accelerate computational algorithm need graphic simd architecture execute single instruction multiple datum time give rise use gpu general computing graph- ic operate system support software like opencl cuda allow- e program advantage gpu windows support use gpu software directx graphic support software call directcompute allow program specify computational kernel high level shader language programming model simd hardware computational kernel run quickly gpu return result main computation run cpu windows 10 native graph- ics stack new windows application use directcompute new version task manager track gpu processor memory usage directx have gpu thread scheduler gpu memory extensibility refer capability operate system advance compute technology facilitate change time devel- oper implement windows layered architecture low level kernel executive run kernel mode provide basic system service abstraction support share use system execu- tive service operate user mode environment subsystem emulate different operate system deprecate today kernel windows use layered

emulate different operate system deprecate today kernel windows use layered architecture loadable driver o system new file system new kind o device new kind networking add system run driver hardware abstraction layer hal windows block diagram limit provide o functionality see pico provider type loadable driver anti malware driver pico providers modular structure system additional operating system support add affect executive figure figure 21.1 show architecture windows 10 kernel subsystem windows use client server model like mach operate system support distribute processing remote procedure call rpcs define open software foundation rpc advantage executive component call advanced local procedure alpc implement highly scalable communication separate process local machine combination tcp ip packet name pipe smb protocol communication process network rpc windows implement distributed common object model dcom infrastructure windows management instrumentation wmi windows remote management winrm mechanism rapidly extend system new service management operate system portable move cpu architecture relatively change windows design portable like unix operating system windows write primarily c c++ relatively little architecture specific source code little assem- bly code port windows new architecture affect windows kernel user mode code windows exclusively write architecture independent port windows kernel architecture- specific code rewrite target cpu conditional compilation need part kernel change major data structure page table format entire windows system recompile new cpu instruction set operate system sensitive cpu architecture cpu support chip hardware boot program cpu support chip collectively know chipset chipset associate boot code determine interrupt deliver describe physical characteristic system provide interface deep aspect cpu architecture error recovery power management burdensome port windows type support chip cpu architecture instead windows isolate chipset dependent code dynamic link library dll call hardware abstraction layer hal load kernel windows kernel depend hal interface underlie chipset detail allow single set kernel driver binary particular cpu different chipset simply load- e

kernel driver binary particular cpu different chipset simply load- e different version hal originally support architecture windows run computer company design market 450 different hals exist time advent standard advanced configuration power interface acpi increas- e similarity component available marketplace merging computer manufacturer lead change today amd64 port windows 10 come single hal interestingly development occur market mobile device today windows support limited number arm chipset appropriate hal code avoid go model multiple hal windows 8 introduce concept hal extensions dll load dynamically hal base detect soc system chip component interrupt controller timer manager dma controller year windows port number different cpu architecture intel ia-32 compatible 32 bit cpu amd64 compatible ia64 64 bit cpu dec alpha dec alpha axp64 mips powerpc cpu cpu architecture fail consumer desktop market windows 7 ship ia-32 amd64 architecture support client computer amd64 server windows 8 32 bit arm add windows 10 support arm64 windows design international multinational use provide support different locale national language support nls api nls api provide specialized routine format date time money accordance national custom string comparison specialize account vary character set unicode windows native character code specifically utl-16le encoding format different linux web standard utf-8 windows support ansi character convert unicode character manipulate 8 bit 16 bit conversion system text string keep resource table inside file replace localize system different language windows vista microsoft ship resource table inside dll mean different executable binary exist different version windows language available single time windows vista multiple user interface mui support multiple locale concurrently important multilingual individual business achieve move resource table separate .mui file live appropriate language directory alongside .dll file support loader pick appropriate file base currently select increase energy efficiency cause battery long laptop internet netbook save significant operating cost power cooling datum center contribute green initiative aim operating cost power cooling datum center contribute green initiative aim lower energy consumption business consumer time windows implement strategy decrease energy use cpu move low power state example lower clock frequency possible addition computer actively windows entire computer low power state sleep save memory secondary storage shut computer hibernation user return computer power continue previous state user need reboot restart application long cpu stay unused energy save computer fast human being lot energy save human think problem program poll wait activity software timer frequently expire keep cpu stay idle long save energy windows 7 extend cpu idle time deliver clock tick interrupt logical cpu 0 currently active cpu skip idle one coalesce eligible software timer small number event server system park entire cpu system heavily load additionally timer expiration distribute single cpu typically charge handle software timer expiration thread run- ning logical cpu 3 cause cpu 3 wake service expiration currently idle nonsleeping cpu handle measure help increase battery life mobile system phone fraction battery capacity laptop windows 8 introduce number feature optimize battery life winrt program model allow precise timer guarantee expiration time timer register new api candidate coalesce unlike win32 timer manually opt concept dynamic tick introduce cpu0 long clock owner active cpu take significantly entire metro modern uwp application model deliver windows store include feature process lifetime manager plm automatically suspend thread process idle second mitigate constant polling behavior application remove ability uwp application background work query gps location force deal system broker efficiently coalesce audio location download request cache datum process suspend finally new component call desktop activity moderator dam windows 8 later version support new type system state call connected standby imagine put computer sleep action take second computer appear disappear hardware turn press button keyboard wake computer take additional second resume phone tablet put

keyboard wake computer take additional second resume phone tablet put device sleep expect second user want screen turn immediately windows merely turn screen program continue run legacy win32 application lack plm timer coalesce continue poll wake screen battery life drain significantly connect standby address problem virtually freeze com- puter power button press screen turn put computer sleep hardware clock stop process service suspend timer expiration delay 30 minute net effect computer run run total state idleness processor peripheral effectively run low power state special hardware firmware require fully support mode example surface brand tablet hardware include capability dynamic device support early history pc industry computer configuration fairly static new device occasionally plug serial printer game port computer step dynamic configuration pc laptop dock pcmciacard device pc quickly connect disconnect set peripheral contemporary pc design enable user plug unplug huge host peripheral frequently support dynamic configuration device continually evolve windows system automatically recognize device plug find install load appropriate driver with- user intervention device unplug driver automatically unload system execution continue disrupt software additionally windows update permit downloading party driver directly microsoft avoid usage installation dvd have user scour manufacturer website peripheral windows server support dynamic hot add hot replace cpu ram dynamic hot remove ram feature allow component add replace remove system interruption limited use physical server technology key dynamic scalability cloud computing especially infrastructure- service iaas cloud computing environment scenario physical machine configure support limited number processor base service fee dynamically upgrade require reboot compatible hypervisor hyper v simple slider owner user interface 21.3

system component architecture windows layered system module operate specific privilege level show early figure 21.1 default privilege level implement processor provide vertical privilege isolation user mode kernel mode windows 10 use hyper v hypervisor provide orthogonal logically independent security model virtual trust levels vtls user enable feature system operate virtual secure mode vsm mode layered privileged system implementation call normal world vtl 0 call secure world vtl 1 world find user mode kernel mode let look structure somewhat detail normal world kernel mode 1 hal extension 2 kernel executive load driver dll dependency user mode collection system process win32 environment subsystem service secure world vsm enable secure kernel executive secure micro hal embed collection isolate trustlets discuss later run secure user mode finally bottommost layer secure world run special processor mode call example vmx root mode intel processor contain hyper v hypervisor component use hardware virtu- alization construct normal secure world boundary user to-kernel boundary provide cpu natively chief advantage type architecture interaction module privilege level keep simple iso- lation need security need necessarily conflate privilege example secure protect component store password unprivileged past operate system designer choose meet isolation need make secure component highly privileged result net loss security system component compromise remainder section describe layer subsystem hypervisor component initialize system vsm enable happen soon user enable hyper v component provide hardware virtualization feature run separate virtual machine provide vtl boundary relate access hardware second level address translation slat functionality discuss shortly hypervisor use cpu specific virtualization extension amd pacifica svmx intel vanderpool vt x intercept interrupt exception memory access instruction port register access choose deny modify redirect effect source destination operation provide hypercall interface enable communicate kernel vtl 0 secure kernel vtl 1 run virtual machine kernel secure kernel secure kernel act kernel mode environment isolate vtl

secure kernel secure kernel act kernel mode environment isolate vtl 1 user- mode trustlet application application implement part windows security model provide system interface kernel interrupt exception attempt enter kernel mode vtl 1 trustlet result enter secure kernel instead secure kernel involve context switching thread scheduling memory man- agement interprocess communication standard kernel task additionally kernel mode driver present vtl 1 attempt reduce attack surface secure world complex implementa- tion remain responsibility normal world component secure kernel act type proxy kernel hand management resource paging scheduling regular kernel service vtl 0 secure world vulnerable denial service attack reasonable tradeoff security design value datum privacy integrity service guarantee addition forward system call secure kernel responsi- bility provide access hardware secret trust platform module tpm code integrity policy capture boot infor- mation trustlets encrypt decrypt datum key normal world obtain sign attest co sign microsoft report integrity token fake replicate outside secure world cpu feature call second level address translation slat secure kernel provide ability allocate virtual memory way physical page back see normal world windows 10 use capability provide additional protection enterprise credential feature call credential guard furthermore device guard mention early activate take advantage vtl 1 capability move digital signature checking secure kernel mean attack software vulner- ability normal kernel force load unsigned driver vtl 1 boundary breach occur device guard protect system kernel mode page vtl 0 authorize execu- tion kernel

ask permission secure kernel secure kernel grant page executable access secure deployment embedded high risk system require level signature validation user mode page additionally work allow special class hardware device usb webcam smartcard reader directly manage user mode driver run vtl 1 umdf framework describe later allow biometric datum securely capture vtl 1 component normal world able intercept currently trustlet allow provide microsoft sign implementation credential guard virtual tpm support new version windows 10 support vsm enclaves allow validly sign necessarily microsoft sign party code wish perform cryptographic calculation software enclave allow regular vtl 0 application enclave run executable code input datum return presumably encrypt output datum information secure kernel https://blogs.technet.micro hal layer software hide hardware chipset difference upper level operate system hal export virtual hard- ware interface kernel dispatcher executive device driver single version device driver require cpu architecture matter support chip present device driver map device access directly chipset specific detail map memory configure o bus set dma cope motherboard specific facility provide hal interface kernel layer windows following main responsibility thread scheduling context switching low level processor synchronization interrupt exception handling switch user mode kernel mode system interface additionally kernel layer imple- ment initial code take boot loader formalize tran- sition windows operate system implement initial code safely crash kernel case unexpected exception assertion inconsistency kernel implement c language assembly language absolutely necessary interface low level hardware architecture direct register access need dispatcher provide foundation executive subsystem dispatcher page memory execution preempt main responsibility thread scheduling context switching implementation synchronization primitive timer management software interrupt asynchronous deferred procedure call interproces- sor interrupt ipi exception dispatching manage hardware software interrupt prioritization system interrupt request level switch user mode kernel mode thread programmer think thread traditional windows actually thread mode execution user mode thread ut kernel- mode thread

thread mode execution user mode thread ut kernel- mode thread kt thread stack ut execution kt ut request system service execute instruction cause trap kernel mode kernel layer run trap handler switch ut stack kt sister change cpu mode kernel thread kt mode complete kernel execution ready switch correspond ut kernel layer call switch ut continue execution user mode kt switch happen windows 7 modify behavior kernel layer support user- mode scheduling uts user mode scheduler windows 7 support cooperative scheduling ut explicitly yield ut call user mode scheduler necessary enter kernel user mode scheduling explain detail section 21.7.3.7 windows dispatcher separate thread run kernel dispatcher code execute kt component ut thread thread go kernel mode circumstance operate system cause kernel thread call circumstance cause kt run dispatcher code operation determine thread run current core like modern operate system windows use thread key schedulable unit executable code process serve container thread process thread thread scheduling state include actual priority processor affinity cpu usage information possible thread state initialize ready deferred- ready standby running waiting transition terminate ready indicate thread wait execute defer ready indicate thread select run specific processor schedule athread run execute processor core run preempt high priority thread terminate allot execution time quantum end wait dispatcher object event signal o completion thread preempt thread different processor place standby state processor mean thread run preemption instantaneous current thread chance finish quantum processor send software interrupt case defer procedure dpc)—to signal processor thread standby state immediately pick execution interestingly thread standby state preempt processor find high priority thread run processor point new high priority thread standby previous thread ready state thread waiting state wait dispatcher object signal thread transition state wait resource necessary execution example wait kernel stack page secondary storage athread enter terminate state finish execution thread begin initialize

storage athread enter terminate state finish execution thread begin initialize state create ready time dispatcher use 32 level priority scheme determine order thread execution priority divide class variable class static class variable class contain thread have priority 1 15 static class contain thread priority range 16 31 dispatcher use link list scheduling priority set list call dispatcher database database use bitmap indicate presence entry list associate priority bit position instead have traverse set list high low find thread ready run dispatcher simply find list associate high bit set prior windows server 2003 dispatcher database global result heavy contention large cpu system windows server 2003 later version global database break apart processor database processor lock new model thread database ideal processor guarantee processor affinity include processor database locate dispatcher simply pick thread list associate high bit set acquire global lock dispatching constant time operation parallelizable cpu single processor system ready thread find dispatcher execute special thread call idle thread role begin transition cpu initial sleep state priority class 0 reserve idle thread multiprocessor system execute idle thread dispatcher look dispatcher database nearby processor take cache topology numa node distance consideration operation require acquire lock processor core order safely inspect list thread steal nearby core dispatcher look near core thread steal processor execute idle thread multiprocessor system cpu idle thread put thread dispatcher database ideal processor cause locality problem imagine cpu execute thread priority 2 cpu bind way cpu execute thread priority 18 cpu bind thread priority 17 ready ideal processor thread cpu thread preempt current run thread ideal processor cpu go ready queue instead wait turn run will happen priority 17 thread give cpu terminate enter wait state windows 7 introduce load balancer algorithm address situation heavy handed disruptive approach locality issue win- dow

late version solve problem nuanced way instead global database windows xp early version processor database windows server 2003 late version new windows version combine approach form share ready queue group processor number cpu form share group depend topology system server client system number choose contention low large processor system avoid locality latency contention issue small client system additionally processor affinity respect processor give group guarantee thread share ready queue appropriate need skip thread keep algorithm constant time windows timer expire 15 millisecond create clock tick examine system state update time housekeeping tick receive thread non idle core interrupt handler run thread kt mode determine thread quantum expire thread time quantum run clock interrupt queue quantum end dpc processor queue dpc result software interrupt processor return normal interrupt priority software interrupt cause thread run dispatcher code kt mode reschedule processor execute ready thread preempt thread priority level round robin fashion thread level ready lower- priority ready thread choose high priority ready thread exist exhaust quantum place situation quantum simply restore default value thread execute windows execute highest- priority ready thread variable priority thread awaken wait operation dispatcher boost priority boost depend type wait associate thread wait o boost depend device thread wait example thread wait sound o large priority increase thread wait disk operation moderate strategy enable o bind thread o device busy permit compute- bind thread use spare cpu cycle background type boost apply thread wait mutex semaphore event synchronization object boost usually hard code value priority level kernel driver option make different change example kernel mode gui code apply boost priority level gui thread wake process window message strategy reduce latency lock notification mechanism signal waiter line execute response state change addition thread associate user active gui window receive priority boost wake reason exist boost enhance response time strategy

priority boost wake reason exist boost enhance response time strategy call foreground priority separation boost tend good response time finally windows server 2003 add lock handoff boost certain class lock critical section boost similar mutex semaphore event boost track ownership instead boost wake thread hard code value priority level boost priority level current owner release lock help situation example thread priority 12 release mutex wait thread priority 8 wait thread receive boost 9 able preempt release thread receive boost 13 preempt instantly acquire critical section thread run boost priority wake wait priority thread lower end quantum long thread base initial priority accord following rule o thread thread boost wake event mutex semaphore priority level lose quantum end thread boost lock handoff boost foreground priority separation boost entire value boost lose thread receive boost type obey rule lose level boost entirety second boost lower thread priority make sure boost apply latency reduction keep o device busy undue execution preference compute- scheduling occur thread enter ready waiting state thread terminate application change thread processor affinity see text thread ready time high priority thread ready low priority thread run low priority thread preempt immediately preemption give high priority thread instant access cpu wait low priority thread quantum complete low priority thread perform event cause operate dispatcher wake wait thread immedi- ately context switch place ready state model essentially distribute scheduling logic dozen win- dow kernel function make currently run thread behave schedule entity contrast operate system rely external scheduler thread trigger periodically base timer advantage windows approach latency reduction cost add overhead inside o state change operation cause current thread perform scheduler work windows hard real time operate system guarantee thread high priority start execute particular time limit guarantee period execution thread block indefinitely dpc interrupt service routine isrs run discuss preempt time

indefinitely dpc interrupt service routine isrs run discuss preempt time high priority thread force round robin thread equal priority quantum end traditionally windows scheduler use sampling measure cpu uti- lization thread system timer fire periodically timer inter- rupt handler take note thread currently schedule execute user kernel mode interrupt occur sam- pling technique originally come cpu high resolution clock clock expensive unreliable access frequently efficient sampling inaccurate lead anomaly charge entire duration clock 15 millisecond cur- rently run thread dpc isr system end completely ignore number millisecond 14.999 spend idle run thread run dpcs isrs combi- nation operation additionally quantum measure base clock tick cause premature round robin selection new thread current thread run fraction start windows vista execution time track hardware timestamp counter tsc include processor pen- tium pro tsc result accurate accounting cpu usage application use note task manager cause scheduler switch thread run quan- tum additionally windows 7 later version track charge tsc isrsand dpc result accurate interrupt time measurement tool use new measurement possible execution time account possible add idle time track tsc accurately compute exact number cpu cycle possible cpu cycle give period fact modern processor dynamically shift frequency result cycle accurate cpu usage measurement tool microsoft sysinternals process explorer use mechanism user interface implementation synchronization primitives windows use number dispatcher object control dispatching synchronization system example object include follow event record event occurrence synchronize occurrence action notification event signal wait thread synchronization event signal single wait thread mutex provide kernel mode user mode mutual exclusion associ- ate notion ownership semaphore act counter gate control number thread access resource thread entity schedule kernel dispatcher associate process encapsulate virtual address space list open resource thread signal thread exit process process exit thread timer track time signal timeout operation long need interrupt periodic activity need schedule like

timeout operation long need interrupt periodic activity need schedule like event timer operate notification mode signal synchronization mode signal dispatcher object access user mode open operation return handle user mode code wait handle synchronize thread operate system interrupt request level irqls hardware software interrupt prioritize service priority order 16 interrupt request level irql windows isa legacy ia-32 use 32 low level irql 0 call passive level default level thread execute kernel user mode level software interrupt level apc dpc level 3 10 represent hardware interrupt base selection pnp manager help hal pci acpi bus driver finally uppermost level reserve clock interrupt quantum management ipi delivery level high level block maskable interrupt typically crash system controlled manner windows irqls define figure 21.2 software interrupt asynchronous deferred procedure call dispatcher implement type software interrupt asynchronous procedure call apcs deferred procedure call dpcs mention early apcs suspend resume exist thread terminate thread deliver notification asynchronous o complete extract modify content cpu register context run thread apcs queue specific thread allow system execute system user code process context user mode execution apc occur arbitrary time thread wait mark alertable kernel mode execution apc contrast instantaneously exe- cut context run thread deliver software interrupt run irql 1 apc level high default irql 0 passive level additionally thread wait kernel mode wait break apc resume apc complete execution type interrupt machine check bus error clock track time traditional pc irq hardware interrupt dispatch defer procedure dpc kernel asynchronous procedure apc interprocessor notification request processor act e.g. dispatch process update tlb windows x86 interrupt request level irlqs dpc postpone interrupt processing handle urgent device interrupt processing isr schedule remain processing queue dpc associate software interrupt run irql 2 dpc level low hardware o interrupt level dpc block device isr addition defer device interrupt processing dispatcher use dpc

device isr addition defer device interrupt processing dispatcher use dpc process timer expiration interrupt current thread execution end scheduling quantum irql 2 high 0 passive 1 apc execution dpc prevent standard thread run current processor keep apc signal completion o.

important dpc routine extended time alternative executive maintain pool worker thread dpc queue work item worker thread execute normal thread schedul- ing irql 0

dispatcher run irql 2 paging operation require wait o involve dispatcher dpc rou- tine restrict page fault pageable system service action result attempt wait dispatcher object signal unlike apcs target thread dpc routine assumption process context processor execute execute context currently execute thread interrupt exception interrupt ipi kernel dispatcher provide trap handling exception interrupt generate hardware software windows define architecture independent exception include integer float point overflow integer float point divide zero illegal instruction data misalignment privileged instruction access violation paging file quota exceed debugger breakpoint trap handler deal hardware level exception call trap elaborate exception handling code perform kernel exception dispatcher exception dispatcher create exception record contain reason exception find exception handler deal exception occur kernel mode exception dispatcher simply call routine locate exception handler handler find fatal system error occur user leave infamous blue screen death signify system failure windows 10 friendly sad face sorrow qr code blue color remain exception handling complex user mode process windows error reporting wer service set alpc error port process win32 environment subsystem set alpc exception port process create detail port section 21.3.5.4 furthermore process debug get debugger port debugger port register exception handler send exception port debugger port find handle exception dispatcher attempt find appropriate exception handler exist contact default unhandled exception handler notify wer process crash crash dump generate send microsoft handler refuse handle exception debugger call catch error debugging debugger run message send process exception port environment subsystem chance react exception finally message send wer error port case unhandled exception handler chance kernel simply terminate process contain thread cause exception wer typically send information microsoft anal- ysis user opt local error report server case microsoft automated analysis able recognize error immediately suggest fix workaround interrupt dispatcher kernel handle interrupt call interrupt service routine isr supply device driver kernel trap-

call interrupt service routine isr supply device driver kernel trap- handler routine interrupt represent interrupt object con- tain information need handle interrupt interrupt object make easy associate interrupt service routine interrupt have access interrupt hardware directly different processor architecture different type number inter- rupt portability interrupt dispatcher map hardware interrupt standard set kernel use interrupt dispatch table bind interrupt level service routine multiprocessor computer windows keep separate interrupt dispatch table idt processor core processor irql set independently mask interrupt interrupt occur level equal irql processor block irql lower kernel level thread isr return interrupt processing windows take advantage property use software inter- rupt deliver apc dpc perform system function synchro- nize thread o completion start thread execution handle windows executive provide set service environment sub- system use good basic overview discuss following service object manager virtual memory manager process manager advanced local procedure facility o manager cache manager security reference monitor plug play power manager registry startup note windows executive include dozen ser- vice total executive organize accord object orient design principle object type windows system define data type set attribute data value set method example function opera- tion help define behavior object instance object type executive perform job set object attribute store datum method perform activity manage kernel mode entity windows use generic set interface manipulate user mode program windows call entity object executive component manipulate object manager example object file registry key device alpc port driver mutexe event process thread see early mutexe process dispatcher object mean thread block dispatcher wait object signal additionally non dispatcher object include internal dispatcher object signal executive service control example file object event object embed signal file modify user mode kernel mode code access object opaque value call handle return api process handle table contain entry track object process system process section

handle table contain entry track object process system process section 21.3.5.11 handle table protect user code kernel mode code manipulate handle handle table windows represent tree structure expand hold 1,024 handle hold 16 million addition handle kernel mode code access object reference pointer obtain call special api handle eventually close avoid keep active reference object similarly kernel code use reference pointer use special api drop reference handle obtain create object open exist object receive duplicated handle inherit handle parent process work issue developer forget close handle open handle process implicitly close exit terminate kernel handle belong system wide handle table driver unload handle automatically close lead resource leak system object manager entity generate object handle natural place centralize call security reference monitor srm section 21.3.5.7 check security attempt open object object manager call srm check process thread right access object access check successful result right encode access mask cache handle table opaque handle represent object kernel identify access grant object important optimization mean file write happen hundred time second security check completely skip handle encode write handle conversely handle read handle attempt write file instantly fail require security check object manager enforce quota maximum memory process use charge process memory occupy reference object refuse allocate memory accumulate charge exceed process quota object reference handle user kernel mode reference pointer kernel mode object manager track count object number handle object number reference handle count number handle refer object handle table include system handle table reference count sum handle count reference plus pointer reference kernel mode component count incremente new pointer need kernel driver decremente component pointer purpose reference count ensure object free reference release datum security descriptor handle close kernel mode component need information object manager maintain windows internal space con- trast unix root system space file system windows use abstract

trast unix root system space file system windows use abstract object manager space visible memory specialized tool debugger instead file system directo- rie hierarchy maintain special kind object call directory object contain hash bucket object include directory object note object name thread object object creator example process mutex want process find acquire inquire state mutex process thread create name reference separate numerical identifier process id pid thread tid object manager support symbolic link space example dos drive letter implement symbolic link global??c symbolic link device object deviceharddiskvolumen represent

mount file system volume object mention early instance object type object type specify instance allocate data field define standard set virtual function object implement standard function implement operation map name object close delete apply security check function specific particular type object implement system service design operate particular object type method specify object type parse function interesting standard object func- tion allow implementation object override default naming behavior object manager use virtual object directo- ries ability useful object internal namespace especially namespace need retain boot o manager file object configuration manager registry key object notable user parse function return windows naming example device object rep- resent file system volume provide parse function allow like global??c:foobar.doc interpret file foobar.doc volume represent device object harddiskvolume2 illustrate naming parse function object handle work look step open file windows 1 application request file name c:foobar.doc open 2 object manager find device object harddiskvolume2 look parse procedure example iopparsedevice object type invoke file relative root file 3 iopparsedevice look file system own volume hard- diskvolume2 call file system look access foobar.doc volume perform internal parsing foo directory find bar.doc file file system allocate file object return o manager parse routine 4 file system return object manager allocate entry file object handle table current process return handle application file successfully open iopparsedevice return error indication application virtual memory manager executive component manage virtual address space physical memory allocation paging memory manager mm design mm assume underlie hardware support virtual physical mapping paging mechanism transparent cache coherence multipro- cessor system allow multiple page table entry map physical page frame mm windows use page base manage- ment scheme base page size support

hardware 4 kb 2 mb 1 gb page datum allocate process physical memory store paging file secondary storage map directly regular file local remote file system page mark zero fill demand initialize page zero map erase previous content 32 bit processor ia-32 arm process 4 gb virtual address space default upper 2 gb identical process windows kernel mode access operate system code datum structure 64 bit architecture amd64 architecture windows provide 256 tb process virtual address space divide 128 tb region user mode kernel mode restriction base hardware limitation soon lift intel announce future processor support 128 pb virtual address space 16 eb availability kernel code process address space important commonly find operate system generally virtual memory map kernel code address space process system execute interrupt receive context switch allow current core run code light weight mapping specificially memory management register need save restore cache invalidate net result fast movement user kernel code compare old architecture kernel memory separate available process address space windows mm use step process allocate virtual memory step reserve page virtual address process virtual address space second step commit allocation assign virtual memory space physical memory space paging file windows limit virtual memory space process consume enforce quota commit memory aprocess de commit memory long free virtual memory space use process api reserve virtual address commit virtual memory handle process object parameter allow process control virtual memory windows implement share memory define section object get handle section object process map memory section range address call view process establish view entire section portion need windows allow section map current process process caller section way section back secondary storage system page file regular file memory map file asection base mean appear virtual address process attempt access section represent physical memory allow 32 bit process access physical memory fit

physical memory allow 32 bit process access physical memory fit virtual address space finally memory protection page section set read read write read write execute execute access copy write let look closely protection setting access page raise exception access exception example check faulty program iterate end array simply detect program attempt access virtual address commit memory user- kernel mode stack use access page guard page detect stack overflow use look heap buffer overrun user- mode memory allocator special kernel allocator device verifier configure map allocation end page follow access page detect programming error access end allocation copy write mechanism enable mm use physical memory efficiently process want independent copy datum section object mm place single share copy virtual memory activate copy write property region memory process try modify datum copy write page mm make private copy page process virtual address translation modern processor use multi- level page table ia-32 operate physical address extension pae mode amd64 processor process page directory contain 512 page directory entry pdes 8 byte size pde point pte table contain 512 page table entry ptes 8 byte size pte point 4 kb page frame physical memory variety reason hardware require page directory pte table level multilevel page table occupy single page number pde pte fit page determine virtual address translate page figure 21.3 diagram structure structure describe far represent 1 gb virtual address translation ia-32

second page directory level need con- taine entry show diagram 64 bit processor entry need amd64 processor fill remain entry second page directory level obtain 512 gb virtual address space support 256 tb require processor need page directory level call pml4 512 entry point low level directory mention early future processor announce intel support 128 pb require fourth page directory level pml5 thank hierarchical mechanism total size page table page need fully represent 32 bit virtual address space process page directory pointer table virtual physical address translation ia-32 8 mb additionally mm allocate page pdes ptes need move page table page secondary storage use actual physical memory overhead paging structure process usually approximately 2 kb page table page fault memory consider virtual address translate physical address ia-32 compatible processor 2 bit value represent value 0 1 2 3 9 bit value represent value 0 511 12 bit value value 0

4,095 12 bit value select byte 4 kb page memory 9 bit value represent 512 pde pte page directory pte table page show figure 21.4 translate virtual address pointer byte address physical memory involve break 32 bit pointer value start significant bit bit index pde level page table select pde contain physical page number page directory page map 1 gb address space bit select pde time second level page directory pde contain physical page number 512 bit select 512 pte select pte table page select pte contain physical page number byte access bit byte offset page physical address byte access construct append low 12 bit virtual address end physical page number find select pte note number bit physical address different number bit virtual address example pae enable mode support windows 8 later version ia-32 mmu extend large 64 bit pte size hardware support 36 bit physical address grant access 64 gb ram single process map address space 4 gb size today amd64 architecture server version windows support large physical address possibly use buy 24 tb late release course time time 4 gb optimistically large physical memory improve performance mm map page directory pte table page contiguous region virtual address process self map allow mm use pointer access current pde pte correspond particular virtual address matter process run self map ia-32 take contiguous 8 mb region kernel virtual address space amd64 self map occupy 512 gb self- map occupy significant address space require additional virtual memory page allow page table page automatically page physical memory creation self map pde level page directory refer page directory page form loop page table translation virtual page access loop take pte table page access loop take low level page directory page access loop take twice forth additional level page directory 64 bit virtual memory translate

forth additional level page directory 64 bit virtual memory translate way virtual address pointer break value amd64 windows use level map 512 page 9 + 9 + 9 + 9 + 12 = 48 bit virtual address avoid overhead translate virtual address look pde pte processor use translation look aside buffer tlb hardware contain associative memory cache map virtual page pte tlb memory management unit mmu processor mmu need walk navigate data structure page table memory need translation miss tlb pdes ptes contain physical page number bit reserve operating system use bit control hardware use memory hardware caching page addition entry specify kind access allow user kernel mode pde mark function pte pde ia-32 11 bit virtual address pointer select pde level translation select pde mark act pte remain 21 bit pointer offset byte result 2 mb size page mix match 4 kb 2 mb page size page table easy operate system significantly improve performance

program improvement result reduce mmu need reload entry tlb pde map 2 mb replace 512 pte map 4 kb new amd64 hardware support 1 gb page operate similar fashion manage physical memory 2 mb page available need difficult continually break 4 kb page cause external fragmentation memory large page result significant internal fragmentation problem typically windows large server application use large page improve performance tlb well suited operating system server application start run system boot memory fragment windows manage physical memory associate physical page seven state free zeroed modified standby bad transition free page available page stale uninitialized content zeroed page free page zero ready immediate use satisfy zero demand fault modified page write process send sec- ondary storage usable process standby page copy information store secondary storage standby page page modify modify page write secondary storage page prefetche expect soon bad page unusable hardware error detect transition page way secondary storage page frame allocate physical memory valid page work set process contain process page table system directly store nonpaged pool valid page contain process page table page state keep separate list accord state type additionally improve performance protect aggressive recycling standby page windows vista later version implement prioritize standby list list construct link corresponding entry page frame number pfn database include entry physical memory page pfn entry include information reference count lock numa information note pfn database represent page phys- ical memory pte represent page virtual memory valid bit pte zero hardware ignore bit mm define use invalid page number state represent bit pte page file page fault mark zero demand page map section object encode pointer appropriate section object pte page write page file contain information locate page secondary storage forth structure page file pte show figure 21.5 t p v bit zero type pte pte include 5

t p v bit zero type pte pte include 5 bit page protection 32 bit page file offset 4 bit select paging file 20 bit reserve additional bookkeeping windows use work set recently lru replacement policy page process appropriate process start assign default minimum work set size point mm start track age page work set working set process allow grow remain physical memory start run low eventually available memory run critically low mm trim work set remove old page age page depend long memory reference mm make determination periodically pass work set process incremente age page mark pte reference pass necessary trim work set mm use heuristic page file page table entry valid bit zero decide trim process remove old page aprocess working set trim plenty memory available give hard limit physical memory use windows 7 later version mm trim process grow rapidly memory plentiful policy change significantly improve responsiveness system process windows track work set user mode process kernel mode region include file cache pageable kernel heap pageable kernel driver code datum working set ts session distinct working set allow mm use different policy trim different category kernel memory mm fault page immediately need research show memory referencing thread tend locality prop- erty page likely adjacent page reference near future think iterate array fetch sequential instruction form executable code thread locality mm fault page fault adjacent page prefetching tend reduce total number page fault allow read cluster improve o performance addition manage commit memory mm manage pro- cess reserve memory virtual address space process asso- ciate tree describe range virtual address use use allow mm fault page table page need pte faulting address uninitialized mm search address process tree virtual address descriptor vads use information fill pte retrieve page case pte table page exist page transparently allocate initialize mm

case pte table page exist page transparently allocate initialize mm case page share section object vad contain pointer section object section object contain information find share virtual page pte initialize point directly start vista windows mm include component call super- fetch component combine user mode service specialized kernel- mode code include file system filter monitor paging operation system second service query trace operation use variety agent monitor application launch fast user switch standby sleep hibernate operation means understand system usage pattern information build statistical model markov chain application user likely launch combination application portion appli- cation example superfetch train understand user launch microsoft outlook morning read e- mail compose e mail later lunch understand outlook background visual studio likely launch text editor go high demand compiler demand little frequently linker frequently documentation code hardly datum superfetch prepopulate standby list make low priority o read secondary storage idle time load think user likely user know fast user switch likely additionally prioritize standby list windows offer prefetche page cache level match statistical likelihood need unlikely be- demand page cheaply quickly evict unexpected need physical memory likely demand soon page keep place long superfetch force system trim work set process touch cache page superfetch monitoring create considerable system overhead mechanical rotational drive seek time millisecond cost balance benefit avoid latency multisecond delay application launch time server system monitoring beneficial give random multiuser workload fact throughput important latency combined latency improvement bandwidth system fast efficient nonvolatile memory ssd monitoring beneficial system situation superfetch disable free spare cpu cycle windows 10 bring large improvement mm introduce component call compression store manager component create compressed store page work set memory compression process type system process shareable page standby list available memory low certain internal algorithm decision page list compress instead evict happen modify page target eviction

page list compress instead evict happen modify page target eviction secondary storage reduce memory pressure avoid write place cause write page compress con- sum page file space take o page today fast multiprocessor system build hardware compression algorithm small cpu penalty highly preferable potential secondary storage windows process manager provide service create delete inter- rogate manage process thread job knowledge parent child relationship process hierarchy group process job hierarchy main- taine process manager involve scheduling thread set priority affinity thread owner process additionally job process manager effect change scheduling attribute throttle ratio quantum val- ue thread thread scheduling proper take place kernel process contain thread process collect large unit call job object original use job object place limit cpu usage work set size processor affinity control multiple process job object man- age large data center machine windows xp later version job object extend provide security relate feature number third- party application google chrome begin job purpose windows 8 massive architectural change allow job influence schedul- ing generic cpu throttling user session aware fairness throttling balancing windows 10 throttling support extend sec- ondary storage o network o additionally windows 8 allow job object nest create hierarchy limit ratio quota system accurately compute additional security power management feature give job object result windows store application uwp application process run job dam introduce early implement connected standby support job finally windows 10 support docker containers key cloud offering use job object call silo job go esoteric data center resource management feature core mechanism process manager multiple windows layered architecture presence environment subsystem process creation complex example process creation win32 environment windows 10 follow note launching uwp modern windows store application call package application appx significantly complex involve factor outside scope discussion 1 win32 application call createprocess 2 anumber parameter conversion behavioral conversion win32

application call createprocess 2 anumber parameter conversion behavioral conversion win32 world nt world 3 createprocess call ntcreateuserprocess api process manager nt executive actually create process initial thread 4 process manager call object manager create process object return object handle win32 call memory manager initialize address space new process handle table key datum structure process environment block pebl contain internal process management datum 5 process manager call object manager create thread object return handle win32 call memory man- ager create thread environment block teb dispatcher initialize scheduling attribute thread set state 6 process manager create initial thread startup context eventually point main routine application ask scheduler mark thread ready immediately suspend put waiting state 7 message send win32 subsystem notify process create subsystem perform additional win32 specific work initialize process compute shutdown level draw animate hourglass donut mouse cursor resumethread api call wake process initial thread control return parent 9 inside initial thread new process user mode link loader take control inside ntdll.dll automatically map process load library dependency dll appli- cation create initial heap set exception handling application compatibility option eventually call main function windows api manipulate virtual memory thread duplicate handle process handle subsystem ser- vice notify process creation perform operation behalf new process have execute directly new process con- text windows support unix fork style process creation anumber feature include process reflectio windows error reporting wer infrastructure process crash windows subsystem linux implementation linux fork api depend debugger support process manager include api suspend resume thread create thread begin suspend mode process manager api set thread register context access process virtual memory thread create current process inject process debugger make use thread injection execute code process debug unfortunately ability allocate manipulate inject memory thread process misuse malicious program run executive thread temporarily attach dif- ferent process thread attach

run executive thread temporarily attach dif- ferent process thread attach kernel worker thread need execute context process originate work request example mm use thread attach need access process work set page table o manager use update status variable process asynchronous o operation facilities client server computing like modern operate system windows use client server model primarily layering mechanism allow put common functionality service equivalent daemon unix term split content parse code pdf reader web browser system action capable code web browser capability save file secondary storage pdf reader ability print document example recent windows 10 operate sys- tem open new york times website microsoft edge browser likely result 12 16 different process complex organization broker renderer parser jitter service client basic server windows computer win32 envi- ronment subsystem server implement operating system personality win32 api inherit windows 95/98 day service user authentication network facility printer spooling web service network file system plug play implement model reduce memory footprint multiple service collect process run svchost.exe program service load dynamic link library dll implement service rely- e user mode thread pool facility share thread wait message section 21.3.5.3

unfortunately pooling originally result poor user experience troubleshoot debug runaway cpu usage memory leak weaken overall security service recent version windows 10 system 2 gb ram dll service run individual svchost.exe process windows recommend paradigm implement client server computing use rpc communicate request inher- ent security serialization service extensibility feature win32 api support microsoft standard dce rpc protocol call ms rpc describe section 21.6.2.7 rpc use multiple transport example name pipe tcp ip implement rpc system rpc occur client server local system alpc transport furthermore rpc heavyweight multiple system- level dependency include winxxiii environment subsystem native windows service kernel directly use alpc available suitable party programmer alpc message pass mechanism similar unix domain socket

party programmer alpc message pass mechanism similar unix domain socket mach ipc server process publish globally visible connection port object client want service server open handle server connection port object send connection request port server accept connection alpc create pair communication- port object provide client connect api handle pair provide server accept api handle pair point message send communication port datagram behave like udp require reply request receive reply client server use synchronous messaging block wait request expect reply asynchronous messaging thread pool mechanism perform work request reply receive need thread block message server locate kernel mode communication port support callback mech- anism allow immediate switch kernel kt user- mode thread ut immediately execute server handler routine alpc message send message passing technique 1 technique suitable small medium sized message 64 kb case port kernel message queue intermedi- eat storage message copy process kernel process disadvantage technique double buffering fact message remain kernel memory intend receiver consume receiver highly contended currently unavailable result megabyte kernel mode memory lock 2 second technique large message case shared- memory section object create port message send port message queue contain message attribute call datum view attribute refer section object receive expose attribute result virtual address mapping section object sharing physical memory avoid need copy large message buffer kernel mode memory sender place datum share section receiver see directly soon consume message possible way implement client server communication exist mailslot pipe socket section object pair event window message use benefit disadvantage rpc alpc remain fully feature safe secure feature rich mechanism communication mechanism vast majority windows process service o manager responsible device driver system implement define communication model allow driver communicate kernel user mode client consumer additionally unix base operating system o target fil object device file system o manager windows allow device driver filter

device file system o manager windows allow device driver filter driver create device stack o flow modify extend enhance original request o manager keep track device driver filter driver load importance file system driver o manager special support implement interface load manage file sys- tem work mm provide memory map file o control windows cache manager handle caching entire o sys- tem o manager fundamentally asynchronous provide synchronous o explicitly wait o operation complete o manager provide model asynchronous o completion include setting event updating status variable call process delivery apc initiate thread use o completion port allow single thread process o completion thread manage buffer o request device driver arrange list device call driver o stack driver represent system driver object single driver operate multiple device driver represent o stack device object contain link driver object additionally nonhardware driver use device object way expose different interface example tcp6 udp6 udp tcp rawip rawip6 device object own tcp ip driver object represent physical device similarly volume secondary storage device object own volume manager driver object handle open device object o manager create file object return file handle instead device handle convert request receive create read write standard form call o request packet irp forward irp driver target o stack processing driver process irp call o manager forward irp driver stack processing finish complete operation represent irp o request complete context different example driver perform o operation force block extended time queue irp worker thread continue process system context original thread driver return status indicate o request pende thread continue execute parallel o operation irp process interrupt service routine complete arbitrary process context final processing need place context initiate o o manager use apc final o completion processing process context originate thread o stack model flexible driver stack build vari- ous

thread o stack model flexible driver stack build vari- ous driver opportunity insert stack filte driver filter driver examine potentially modify o opera- tion volume snapshotting shadow copy disk encryption bitlocker build example functionality implement filter driver execute volume manager driver stack file system filter driver execute file system implement func- tionalitie hierarchical storage management single instancing file remote boot dynamic format conversion party use file- system filter driver implement anti malware tool large number file system filter windows server 2003 later version include filte manager component act sole file system filter load minifilter order specific altitude relative priority model allow filter transparently cache datum repeat query have know request provide strict load ordering device driver windows write windows driver model wdm specification model lay requirement device driver include layer filter driver share common code han- dling power plug play request build correct cancellation logic richness wdm write wdm device driver new hardware device involve great deal work case port miniport model make unnecessary certain hardware device range device require similar processing audio driver storage controller ethernet controller instance device share common driver class call port driver port driver implement standard operation class call device specific routine device miniport driver implement device specific function- ality physical link layer network stack implement way ndis.sys port driver implement generic network processing functionality call network miniport driver specific hardware command relate send receive network frame ethernet similarly wdm include class miniclass model certain class device implement generic way single class driver callout miniclass specific hardware functionality example windows disk driver class driver driver cd dvd tape drive keyboard mouse driver class driver type device need miniclass battery class driver example require miniclass external uninterruptible power supply upss sell vendor port miniport class miniclass model significant kernel face code write model useful custom hardware

model significant kernel face code write model useful custom hardware logical nonhardware driver start windows 2000 service pack 4 kernel mode driver write kernel mode driver framework kmdf provide simplified programming model driver wdm option user mode driver framework umdf allow driver write user mode reflecto driver kernel forward request kernel o stack framework windows driver foundation model reach version 2.1 windows 10 contain fully compatible api kmdf umdf fully open source driver need operate kernel mode easy develop deploy driver user mode umdf strongly recommend new driver make system reliable failure user mode driver cause kernel system crash operate system caching block device system usually physical block level instead windows provide centralized caching facility operate logical virtual file level cache manager work closely mm provide cache service component control o manager mean cache operate remote file network share logical file custom file system size cache change dynamically accord free memory available system grow large 2 tb 64 bit system cache manager maintain private working set share system process working set allow trimming page cache file effectively build cache cache manager memory map file kernel memory use special interface mm fault page trim private working set let advantage additional caching facility provide memory manager cache divide block 256 kb cache block hold view memory map region file cache block describe virtual address control block vacb store virtual address file offset view number process view vacbs reside array maintain cache manager array critical low priority cache datum improve performance situation memory pressure o manager receive file user level read request o manager send irp o stack volume file reside file mark cacheable file system call cache manager look request datum cache file view cache manager calculate entry file vacb index array correspond byte offset request entry point view cache invalid invalid

correspond byte offset request entry point view cache invalid invalid cache manager allocate cache block corresponding entry vacb array map view cache block cache manager attempt copy datum map file caller buffer copy succeed operation complete copy fail page fault cause mm send noncached read request o manager o manager send request driver stack time request paging operation bypass cache manager read datum file directly page allocate cache manager completion vacb set point page datum cache copy caller buffer original o request complete figure 21.6 show overview possible synchronous operation cache file o handle fast o mechanism mechanism parallel normal irp base o call driver stack directly pass irp save memory time irp involve operation block extended period time queue worker thread operation reach file system call cache manager operation fail information cache o manager attempt operation normal irp path akernel level read operation similar datum access directly cache copy buffer user space use file system metadata data structure describe file system kernel use cache manager mapping interface read metadata modify metadata file system use cache manager pin interface pin page lock page physical memory page frame mm manager page page update metadata file system ask cache manager unpin page amodifie page mark dirty mm flush page secondary storage improve performance cache manager keep small history read request history attempt predict future request cache manager find pattern previous request sequential access forward backward prefetche datum cache request submit application way application find datum cache need wait secondary storage o. cache manager responsible tell mm flush content cache cache manager default behavior write caching accumulate write 4 5 second wake cache- writer thread write caching need process set flag open file explicit cache flush function fast write process potentially fill free cache page cache writer thread chance wake flush page sec-

cache page cache writer thread chance wake flush page sec- ondary storage cache writer prevent process flood system following way free cache memory low cache manager temporarily block process attempt write datum wake cache writer thread flush page secondary storage fast- write process actually network redirector network file system block long cause network transfer time retransmit retransmission waste network bandwidth pre- vent waste network redirector instruct cache manager limit backlog write cache network file system need datum secondary storage network interface cache manager provide dma interface datum directly move datum directly avoid need copy datum intermediate buffer security reference monitor centralize management system entity object manager enable windows use uniform mechanism perform run time access validation audit check user accessible entity system additionally entity manage object manager access api routine perform security check thread open handle protect data structure object security reference monitor srm check effective security token object security descriptor contain access control list discretionary access control list dacl system access control list sacl)—to process necessary access right effective security token typically token thread process token thread process associate security token login process lsass.exe authenticate user security token attach user process userinit.exe copy child process token contain security identity sid user sid group user belong privilege user integrity level process attribute claim associate user relevant capability default thread explicit token cause share common token process mechanism call imperson- ation thread run process security token belong user set thread specific token belong user impersonate user point effective token token thread operation quota limitation subject user token thread later choose revert old identity remove thread specific token effective token process impersonation facility fundamental client server model service act behalf variety client different secu- rity id right impersonate user deliver connection client process server process impersonation allow server access system service client

process server process impersonation allow server access system service client order access create object file behalf client server process trustworthy carefully write robust attack client server process impersonate user subsequent client request windows provide api support impersonation alpc rpc dcom layer name pipe layer srm responsible manipulate privilege security token special privilege require user change system time load driver change firmware environment variable additionally certain user powerful privilege override default access control rule include user perform backup restore operation file system allow bypass read write restriction debug process allow bypass security feature forth integrity level code execute process represent token integrity level type mandatory labeling mechanism men- tione early default process modify object integrity level high code execute process per- mission grant addition read process object high integrity level object protect read access manually change mandatory policy associate secu- rity descriptor inside object file process integrity level store sacl distinguish typical discretionary user group permission store dacl integrity level introduce hard code system attack external content parse software like browser pdf reader software expect run low integrity level example microsoft edge run low integrity adobe reader google chrome regular application microsoft word run medium integrity finally expect application run adminis- trator setup program run high integrity create application run low integrity level place burden developer implement security feature create client server model support broker parser renderer mention early order streamline security model windows 8 introduce applica- tion container call appcontainer special extension token object run appcontainer application automatically process token adjust following way 1 token integrity level set low mean application write modify object file key process system read process system 2 group user sid disable ignore token let application launch user anne belong world group file accessible anne world inaccessible application 3 privilege handful remove token prevent powerful system

inaccessible application 3 privilege handful remove token prevent powerful system call system wide operation permit 4 special appcontainer sid add token correspond sha-256 hash application package identifier valid security identifier token object wish directly accessible application need explicitly appcontainer sid read write access 5 set capability sid add token base application manifest file application instal capability show user agree application appcontainer mechanism change security model discretionary system access protect resource define user group mandatory system application unique security identity access occur application basis separation privilege permission great leap forward security place potential burden resource access capability broker help alleviate burden capability system broker implement windows per- form action behalf package application example assume harold package application access harold file system harold sid disable situation broker check play user media capability allow music player process read mp3 file locate harold music directory harold force mark file appcontainer sid favorite medium player application long application play user media capability harold agree download application final responsibility srm log security audit event iso standard common criteria international successor orange book standard develop united states department defense require secure system ability detect log attempt access system resource easily trace attempt unauthorized access srm responsible make access check generate audit record write lsass.exe security event operate system use plug play pnp manager recognize adapt change hardware configuration pnp device use standard pro- tocol identify system pnp manager automatically recognize instal device detect change device system oper- ate manager keep track hardware resource device potential resource take care load appropriate driver management hardware resource primarily interrupt dma channel o memory range goal determin- e hardware configuration device able operate success- fully pnp manager windows driver model driver bus driver detect enumerate device bus pci usb function driver implement functionality particular device bus pnp manager handle dynamic reconfiguration

implement functionality particular device bus pnp manager handle dynamic reconfiguration follow get list device bus driver load driver send add device request appropriate driver device work tandem special resource arbiter own bus driver pnp manager figure optimal resource assignment send start device request driver specify resource assignment related device device need reconfigure pnp manager send query stop request ask driver device temporarily disable driver disable device pende operation complete new operation prevent start finally pnp manager send stop request reconfigure device new start device request pnp manager support request example query- remove operate similarly query stop employ user get ready eject removable device usb storage device surprise remove request device fail user remove device tell system stop finally remove request tell driver stop device permanently program system interested addition removal device pnp manager support notification notification example give file manager information need update list secondary storage volume new storage device attach remove instal device result start new service system previously service frequently set run system boot continue run associate device plug system run order receive pnp notification windows 7 introduce service trigger mechanism service control manager scm services.exe manage system service mechanism service register start scm receive notification pnp manager device interest add system windows work hardware implement sophisticated strategy energy efficiency describe section 21.2.8 policy drive strategy implement power manager power manager detect current system condition load cpu o device improve energy efficiency reduce performance responsiveness system need low power manager entire system efficient sleep mode write content memory secondary storage turn power allow system primary advantage sleep system enter state fairly quickly second lid close laptop return sleep fairly quick power turn low level cpu o device memory continue power content lose note early mobile device second add unreasonable user experience power manager work desktop activity moderator kick

unreasonable user experience power manager work desktop activity moderator kick connected standby state soon screen turn connected standby virtually freeze computer computer sleep hibernation take considerably long enter sleep entire content memory transfer secondary storage system turn fact system fact turn significant advantage loss power system battery swap laptop desktop system unplug save system datum lose unlike shutdown hibernation save currently run system user resume leave furthermore hibernation require power system remain hiberna- tion indefinitely feature extremely useful desktop server system laptop battery hit critical level put system sleep battery low result loss datum battery run power sleep state windows 7 power manager include processor power man- ager ppm specifically implement strategy core parking cpu throttling boosting addition windows 8 introduce power framework pofx work function driver implement specific functional power state mean device expose inter- nal power management clock speed current power draw forth system use information fine grained control device example instead simply turn device system turn specific component like pnp manager power manager provide notification rest system change power state application want know system shut start save state secondary storage mention early dam need know screen turn windows keep configuration information internal repository datum call hive manage windows configuration manager commonly know registry configuration manager implement component executive separate hive system information user preference software information security boot option additionally new application security model introduce appcontainers uwpmodern metro package application windows 8 applica- tion separate hive call application hive registry represent configuration state hive hierarchical namespace key directory contain set arbitrarily sized value win32 api value specific type unicode string 32 bit integer untyped binary datum registry treat value leave high api layer infer structure base type size example prevent 32 bit integer 999 byte unicode string theory new key value create initialize new software instal modify

theory new key value create initialize new software instal modify reflect change configuration software practice registry general purpose database interprocess communication mechanism inventive purpose restart application system time configuration change nuisance instead program rely kind notification provide pnp power man- ager learn change system configuration registry supply notification thread register notify change registry thread detect adapt configuration change record registry furthermore registry key object manage object manager expose event object dispatcher allow thread waiting state associate event configuration manager signal key value modify significant change system update operate system driver instal danger configuration datum corrupted example work driver replace nonworking driver application fail install correctly leave partial information registry windows create system restore point make change restore point contain copy hive change return version hive order corrupted system work improve stability registry configuration registry implement variety self heal algorithm detect fix certain case registry corruption additionally registry internally use phase commit transactional algorithm prevent corruption individual key value update mechanism guarantee integrity small portion registry individual key value supplant system restore facility recover damage registry configuration cause failure software booting windows pc begin hardware power firmware begin execute rom old machine firmware know bios modern system use uefi unified extensible firmware interface fast modern make well use facility contemporary processor additionally uefi include feature call secure boot provide integrity check digital signature verification firmware boot time component digital signature check guarantee microsoft boot time component vendor firmware present boot time prevent early party code firmware run power self test post diagnostic identify device attach system initialize clean power state build description acpi firmware find system boot device load windows boot manager program boot- mgfw.efi uefi system begin execute machine hibernate winresume.efi program load restore run system secondary storage system continue execution point reach right hibernat- ing machine shut bootmgfw.efi perform

execution point reach right hibernat- ing machine shut bootmgfw.efi perform initialization system load winload.efi program load hal.dll kernel ntoskrnl.exe dependency driver need booting system hive winload transfer execution procedure somewhat different windows 10 system vir- tual secure mode enable hypervisor turn win- load.efi instead load hvloader.exe hvloader.dll initialize hypervisor intel system hvix64.exe amd system use hvax64.exe hypervisor set vtl 1 secure world vtl 0 normal world return winload.efi load secure kernel securekernel.exe dependency secure kernel entry point call initialize vtl 1 return loader vtl 0 resume step describe kernel initialize create process idle process serve container idle thread system wide cpu idle time easily compute system process contain internal kernel worker thread system thread create driver polling house- keeping background work memory compression process new windows 10 work set compose compressed standby page store manager alleviate system pressure optimize paging finally vsm enable secure system process represent fact secure kernel load user mode process create kernel ses- sion manager subsystem smss similar init initialization process unix smss perform initialization system includ- ing establish paging file create initial user session session represent log user session 0 run system wide background process lsass service session give instance smss process exit ses- sion create session ephemeral smss load win32 environment subsystem csrss.exe driver win32k.sys session 0

smss run winlogon process launch logonui process capture user credential order lsass log user launch userinit explorer process implement windows shell start menu desktop tray icon notification center following list itemize aspect booting smss complete system initialization start smss ses- sion 0 smss login session 1 wininit run session 0 initialize user mode start lsass lsass security subsystem implement facility authentication user user credential protect vsmthrough credential guard lsaiso bioiso start vtl 1 trustlets lsass service contain service control manager scm supervise background activity system include user mode service number service register start system boot start demand trigger event arrival device csrss win32 environment subsystem process start session mainly handle mouse keyboard input need separate user winlogon run windows session session 0 log user launch logonui present logon

user interface start windows xp system optimize boot process prefetche page file secondary storage base previous boot system disk access pattern boot lay system file disk reduce number o operation require windows 7 reduce process necessary start system allow service start need system start windows 8 reduce boot time parallelize driver load pool worker thread pnp subsystem support uefi boot time transition efficient approach contribute dramatic reduction system boot time eventually little improvement possible address boot time concern especially mobile system ram core limit windows 8 introduce hybrid boot feature combine hibernation simple logoff current user user shut system application session exit system return logonui prompt hibernate system turn resume quickly logon screen give driver chance reinitialize device give appearance boot work occur 21.4 terminal service fast user switching windows support gui base console interface user key- board mouse display system support audio video example audio input cortana windows voice recognition vir- tual assistant software power machine learning cortana make system convenient increase accessibility user motor disability windows 7 add support multi touch hardware allow user input datum touch screen finger video input capability accessibility security windows hello security feature advanced 3d heat sense face map camera sensor uniquely identify user requir- e traditional credential new version windows 10 eye motion sens- e hardware mouse input replace information posi- tion gaze eyeball accessibility future input experience likely evolve microsoft hololens augmented reality pc course envision personal computer inherently single user machine time windows support sharing pc multiple user user log gui session create represent gui environment contain process necessary run application windows allow multiple session exist time single machine client version windows support single console consist monitor keyboard mouse connect pc session connect console time logon screen display console user create new session attach exist session allow multiple user share

create new session attach exist session allow multiple user share single pc have log user microsoft call use session fast user switching macos similar feature user pc create new session connect exist session computer remote desktop terminal service feature ts make connection protocol call remote desktop protocol rdp user employ feature connect session work pc home pc remote desktop remote troubleshooting scenario remote user invite share session user log session console remote user watch user action give control desktop help resolve computing problem use terminal service use mirror feature alternative user share session instead create separate corporation use corporate system maintain datum center run user session access corporate resource allow user access resource pc exclusively dedicate machine terminal server server computer handle hundred remote desktop session form thin client computing individual computer rely server function rely data- center terminal server improve reliability manageability security corporate compute resource 21.5 file system native file system windows ntfs local volume associate usb thumb drive flash memory camera external storage device format 32 bit fat file system portability fat old file system format understand system windows software run camera disadvantage fat file system restrict file access authorized user solution secure datum fat run application encrypt datum store file system contrast ntfs use acl control access individual file sup- port implicit encryption individual file entire volume windows bitlocker feature ntfs implement feature include data recovery fault tolerance large file file system multiple data stream unicode name sparse file journaling volume shadow copy ntfs internal layout fundamental entity ntfs volume volume create win- dows logical disk management utility base logical disk partition volume occupy portion device entire device span device volume manager protect content volume level raid ntfs deal individual sector storage device instead use cluster unit storage allocation cluster size power 2 configure ntfs file

unit storage allocation cluster size power 2 configure ntfs file system format default cluster size base volume size—4 kb volume large 2 gb give size today storage device sense use cluster size large windows default achieve well performance performance gain come expense internal fragmenta- ntfs use logical cluster number lcn storage address assign number cluster beginning device end scheme system calculate physical storage offset byte multiply lcn cluster size file ntfs simple byte stream unix structured object consist type attribute attribute file independent byte stream create delete read write attribute type standard file include file name file alias ms dos short creation time security descriptor specify access control list user datum store traditional data file unnamed data attribute contain file datum additional data stream create explicit name iprop interface component object model discuss later chapter use name datum stream store property ordinary file include thumbnail image general attribute add nec- essary access file attribute syntax ntfs return size unnamed attribute response file query operation run dir command file ntfs describe record array store special file call master file table mft size record determine file system create range 1 4 kb small attribute store mft record call resident attribute large attribute unnamed bulk datum call nonresident attribute store contiguous extent device pointer extent store mft record small file datum attribute fit inside mft record file attribute highly fragment pointer need point fragment record mft large case file describe record call base fil record contain pointer overflow record hold additional pointer attribute file ntfs volume unique id call fil reference file reference 64 bit quantity consist 48 bit file number 16 bit sequence number file number record number array slot mft describe file sequence number incremente time mft entry reuse sequence number enable ntfs perform internal consistency check catch stale reference delete

enable ntfs perform internal consistency check catch stale reference delete file mft entry reuse new file ntfs b+ tree unix ntfs namespace organize hierarchy directory directory use data structure call b+ tree store index file name directory b+ tree length path root tree leaf cost reorganize tree eliminate index root directory contain level b+ tree large directory level contain pointer disk extent hold remainder tree entry directory contain file reference file copy update timestamp file size take file resident attribute mft copy information store directory directory listing efficiently generate file name size update time available directory need gather attribute mft entry file ntfs volume metadata store file file mft second file recovery mft damage contain copy 16 entry mft file special purpose include following file log file record metadata update file system volume file contain volume version ntfs format volume bit tell volume corrupt need check consistency chkdsk attribute definitio table indicate attribute type volume operation perform root directory level directory file system hierarchy bitmap fil indicate cluster volume allocate file free boot fil contain startup code windows locate particular secondary storage device address find easily simple rom bootstrap loader boot file contain physical address mft bad cluster file keep track bad area volume ntfs use record error recovery keep ntfs metadata actual file useful property discuss section 21.3.5.6 cache manager cache file datum ntfs metadata reside file datum cache mechanism ordinary datum simple file system power failure wrong time damage file system datum structure severely entire volume scramble unix file system include ufs zfs store redundant metadata storage device recover crash fsck pro- gram check file system datum structure restore forcibly consistent state restore involve delete damage file free data cluster write user datum properly record file system metadata structure check slow process result loss significant amount datum ntfs

structure check slow process result loss significant amount datum ntfs take different approach file system robustness ntfs file- system data structure update perform inside transaction data structure alter transaction write log record contain redo undo information datum structure change transaction write commit record log signify transaction succeed crash system restore file system datum structure commit transaction sure change reach file system commit successfully crash periodically usually 5 second checkpoint record write log system need log record prior checkpoint recover crash discard log file grow bound time system startup ntfs volume access ntfs automatically perform file system recovery scheme guarantee user file content correct crash ensure file system datum structure metadata file undamaged reflect consistent state exist prior crash possible extend transaction scheme cover user file microsoft take step windows vista log store metadata file beginning volume create fix maximum size file system format section logging area circular queue log record restart area hold context information position logging area ntfs start read recovery fact restart area hold copy information recovery possible copy damage crash logging functionality provide log file service addition write log record perform recovery action log file service keep track free space log file free space get low log- file service queue pende transaction ntfs halt new o operation progress operation complete ntfs call cache manager flush datum reset log file perform queue transaction security ntfs volume derive windows object model ntfs file reference security descriptor specify owner file access control list contain access permission grant deny user group list early version ntfs separate security descriptor attribute file begin windows 2000 security descriptor attribute point share copy significant saving storage space caching space file identical normal operation ntfs enforce permission traversal directory file path name compatibility posix check enable option inherently expensive modern parsing file path name use prefix

option inherently expensive modern parsing file path name use prefix matching directory-directory parsing path name prefix matching algorithm look string cache find entry long match example entry foobardir match foobardir2dir3myfile prefix match cache
allow path traversal begin deeply tree save step enforce traversal check mean user access check directory level instance user lack permission traverse foobar start access foobardir error ntfs perform datum compression individual file data file directory compress file ntfs divide file datum compres- sion unit block 16 contiguous cluster compression unit write data compression algorithm apply result fit few 16 cluster compress version store read ntfs determine datum compress length store compression unit 16 cluster improve per- formance read contiguous compression unit ntfs prefetche decompress ahead application request sparse file file contain zero ntfs use tech- nique save space cluster contain zero write actually allocate store storage device instead gap leave sequence virtual cluster number store mft entry file read file ntfs find gap virtual cluster num- ber zero fill portion caller buffer technique unix mount points symbolic links hard links mount point form symbolic link specific directory ntfs introduce windows 2000 provide mechanism organize storage

volume flexible use global name like drive letter mount point implement symbolic link associate datum contain true volume ultimately mount point supplant drive letter completely long transition dependence application drive letter scheme windows vista introduce support general form symbolic link similar find unix link absolute relative point object exist point file directory volume ntfs support hard link single file entry directory volume ntfs keep journal describe change file system user mode service receive notification change journal identify file change read journal search indexer service use change journal identify file need index file replication service use identify file need replicate network volume shadow copies windows implement capability bring volume known state

volume shadow copies windows implement capability bring volume known state create shadow copy consistent view volume technique know snapshot file system make shadow copy volume form copy write block modify shadow copy create store original form copy achieve consistent state volume require cooperation application system know datum application stable state application safely server version windows use shadow copy efficiently maintain old version file store file server allow user document exist early point time user recover file accidentally delete simply look previous version file pull backup medium windows support peer peer client server networking facility network management networking component win- dow provide data transport interprocess communication file sharing network ability send print job remote printer describe networking windows mention internal networking interface network device interface specificatio ndis transport driver interface tdi ndis interface develop 1989 microsoft 3com separate network adapter transport protocol change affect ndis reside interface data link network layer iso model enable protocol operate different network adapter term iso model tdi interface transport layer layer 4 session layer layer 5 interface enable session layer component use available transport mechanism similar reasoning lead stream mechanism unix tdi support connection base connectionless transport function send type datum windows implement transport protocol driver driver load unload system dynamically practice system typically reboot change windows come networking protocol discuss number protocol server message block server message block smb protocol introduce ms dos 3.1 system use protocol send o request network smb protocol message type session control message command start end redirector connection shared resource server redirector use file message access file server printer message send datum remote print queue receive status information queue message message communicate workstation version smb protocol publish common internet file system cifs support number transmission control protocol internet protocol transmission control protocol internet protocol tcp ip suite internet de

transmission control protocol internet protocol tcp ip suite internet de facto standard networking infrastructure windows use tcp ip connect wide variety operating system hardware platform windows tcp ip package include simple network management protocol snmp dynamic host configuration pro- tocol dhcp old windows internet service wins windows vista introduce new implementation tcp ip support ipv4 ipv6 network stack new implementation support offloading network stack advanced hardware achieve high performance server windows provide software firewall limit tcp port program network communication network firewall com- monly implement router important security measure have firewall build operating system make hardware router unnecessary provide integrated management easy use point point tunneling protocol point point tunneling protocol pptp protocol provide win- dows communicate remote access server module run win- dows server machine client system connect internet remote access server encrypt datum send connec- tion support multiprotocol virtual private network vpns http protocol information world wide web windows implement http kernel mode driver web server operate low overhead connection networking stack http fairly general protocol windows make available transport option web distribute authoring versioning protocol web distribute authoring versioning webdav http base proto- col collaborative authoring network windows build webdav redirector file system build directly file system enable webdav work file system feature encryption personal file store

securely public place webdav use http protocol windows cache file locally pro- grams use read write operation part file name pipe connection orient messaging mechanism process use name pipe communicate process machine name pipe access file system interface security mechanism file object apply name pipe smb protocol support name pipe name pipe communication process different system format pipe name follow uniform naming convention unc unc look like typical remote file format server nameshare namexyz
server identify server network share identify resource available network user directory file name pipe printer xyz

resource available network user directory file name pipe printer xyz normal file path
remote procedure calls remote procedure call rpcs mention early client server mecha- nism enable application machine procedure code machine client call local procedure stub routine pack argument message send network particular server process client stub routine block mean- server unpack message call procedure pack return result message send client stub client stub unblock receive message unpack result rpc return caller packing argument call marshaling client stub code descriptor necessary pack unpack argu- ment rpc compile specification write microsoft interface definitio language windows rpc mechanism follow widely distributed- computing environment standard rpc message program write use windows rpcs highly portable rpc standard detail hide architectural difference computer size binary number order byte bit computer word specify standard data format rpc message component object model component object model com mechanism interprocess com- munication develop windows com object provide well- define interface manipulate datum object instance com infrastructure microsoft object linking embedding ole technology insert spreadsheet microsoft word document windows service provide com interface addition distribute extension call dcom network utilize rpc provide transparent method develop distribute application redirector server windows application use windows o api access file remote computer local provide remote com- puter run cifs server provide windows aredirector

client object forward o request remote system satisfy server performance security redirector server run kernel mode detail access remote file occur follow 1 application call o manager request file open file standard unc format 2 o manager build o request packet describe section 3 o manager recognize access remote file call driver call multiple unc provider mup 4 mup send o request packet asynchronously register 5 redirector satisfy request respond mup avoid ask redirector question future mup use cache remember redirector handle file 6 redirector send network request remote

remember redirector handle file 6 redirector send network request remote system 7 remote system network driver receive request pass 8 server driver hand request proper local file system driver 9 proper device driver call access datum 10 result return server driver send datum request redirector redirector return datum call application o manager asimilar process occur application use win32 network api unc service module call multi provider router instead mup portability redirector server use tdi api network trans- port request express high level protocol default smb protocol describe section 21.6.2 list redirector maintain system hive registry distributed file system unc name convenient multiple file server available serve content unc name explicitly include server windows support distribute file syste dfs protocol allow network administrator serve file multiple server single distribute space folder redirection client caching improve pc experience user frequently switch com- puter windows allow administrator user roam profile user preference setting server folder redirection automatically store user document file server work computer long attach network user take laptop airplane user line access redirect file windows use client caching csc csc computer line copy server file local machine well performance file push server change computer disconnected file available update server defer time computer online networked environment natural group user student computer laboratory school employee department busi- ness frequently want member group able access share resource computer group manage global access right group windows use concept domain pre- viously domain relationship whatsoever domain system dns map internet host name ip address closely relate specifically windows domain group windows workstation server share common security policy user database windows use kerberos protocol trust authentication windows domain thing kerberos realm windows use hierarchical approach establish trust relationship related domain trust rela- tionship base dns allow transitive trust flow hierarchy approach reduce number trust require n domain

trust flow hierarchy approach reduce number trust require n domain n (n 1 o(n workstation domain trust domain controller correct information access right user load user access token lsaas user retain ability restrict access workstation matter domain controller active directory windows implementation lightweight directory- access protocol ldap service active directory store topology infor- mation domain keep domain base user group account password provide domain base store windows feature need windows group policy administrator use group policy establish uniform standard desktop preference software corporate information technology group uniformity drastically reduce cost computing 21.7 programmer interface win32 api fundamental interface capability windows section describe main aspect win32 api access kernel object sharing object process process management interprocess com- munication memory management access kernel object windows kernel provide service application program use application program obtain service manipulate kernel object process gain access kernel object name xxx call createxxx function open handle instance xxx handle unique process depend object open create function fail return 0 return special constant name invalid handle value process close handle call security attribute sa sa.nlength = sizeof(sa sa.lpsecuritydescriptor = null sa.binherithandle = true handle hsemaphore = createsemaphore(&sa 1 1 null wchar wszcommandline[max path

l"another process.exe d hsemaphore createprocess(l"another process.exe wszcommandline null null true code enable child share object inherit handle closehandle function system delete object count handle reference object process drop zero share object process windows provide way share object process way child process inherit handle object parent call createxxx function parent supply security attribute structure binherithandle field set true field create inheritable handle child process create pass value true createprocess function binherithandle argument figure 21.7 show code sample create semaphore handle inherit child assume child process know handle share parent child achieve interprocess communication share object example figure 21.7 child process get value handle command line argument share semaphore parent process second way share object process object object create second process open method drawback windows provide way check object choose exist object space global regard object type instance application create share single object name foo distinct object possibly different type desire name object advantage unrelated process readily share process call createxxx function supply parameter second process get handle share object call openxxx createxxx show example figure 21.8 way share object duplicatehandle function method require method interprocess communication pass duplicate handle give handle process value handle process second process handle object share example method show figure 21.9 process handle hsemaphore = createsemaphore(null 1 1 l"mysem1 process b handle hsemaphore = opensemaphore(semaphore access code share object lookup windows process load instance application thread executable unit code schedule kernel dispatcher process contain thread process create thread process call createprocess api routine load dynamic link library process create initial thread process additional thread create createthread function process want process b access semaphore process dword dwprocessbid ipc mechanism handle hsemaphore = createsemaphore(null 1 1 null handle hprocess = openprocess(process dup handle false 0 false

duplicate access send value semaphore process b message share memory object process b handle hsemaphore = value semaphore message use hsemaphore access semaphore code share object pass handle thread create stack default 1 mb specify argument createthread priority win32 environment base native kernel nt scheduling model priority value choose win32 api use priority class 1 idle priority class nt priority level 4 2 normal priority class nt priority level 6 3 normal priority class nt priority level 8) 4 normal priority class nt priority level 10 5 high priority class nt priority level 13 6

realtime priority class nt priority level 24 process typically member normal priority class parent process idle priority class class specify createprocess call priority class process default thread execute process change setpriorityclass function pass argument start command user increase scheduling priority privilege process realtime priority class administrator power user privilege default user switch interactive process workload system need schedule appropriate thread provide good responsiveness lead short quantum execution user choose particular process good throughput particular process expect reason windows special scheduling rule process realtime priority class windows distinguish process associate foreground window screen background process process move foreground windows increase scheduling quantum thread factor 3 cpu bind thread foreground process run time long similar thread background process server system operate large quantum client system factor 6 behavior enable server system type system scheduling parameter customize appropriate system dialog registry key thread start initial priority determine class priority alter setthreadpriority function function take argument specify priority relative base priority class thread priority lowest base 2 thread priority normal base 1 thread priority normal base + 0 thread priority normal base + 1 thread priority highest base + 2 designation adjust priority recall section 21.3.4.3 kernel priority class 16–31 static class 1–15 variable class thread priority idle set pri- ority 16 static priority thread 1 variable priority thread thread priority time critical set priority 31 real time thread 15 variable priority thread kernel adjust priority variable class thread dynamically depend thread o bind cpu bind win32 api provide method disable adjustment setprocesspriority- boost setthreadpriorityboost function thread suspend resume thread create suspend state place suspend state later use suspendthread function suspend thread schedule kernel dispatcher move sus- pende state use resumethread function function set counter thread suspend twice resume twice run synchronize concurrent access share object thread kernel pro- vide synchronization object semaphore mutexe

share object thread kernel pro- vide synchronization object semaphore mutexe dis- patcher object discuss section 21.3.4.3 thread synchronize kernel service operate kernel object thread process file dispatcher object synchronization ker- nel dispatcher object achieve use waitforsingleobject waitformultipleobjects function function wait dispatcher object signal method synchronization available thread process want execute code exclusively win32 critical section object user mode mutex object acquire release enter kernel multiprocessor win32 critical section attempt spin wait critical section hold thread release spinning take long acquire thread allocate kernel mutex yield cpu critical section particularly efficient kernel mutex allocate contention attempt spin mutexe program actually contend saving significant critical section thread process initializecriticalsection thread want acquire mutex call entercriticalsection later call leavecritical- section release mutex tryentercriticalsection function attempt acquire mutex block program want user mode reader writer lock mutexe win32 support slim reader writer srw lock srw lock api similar critical section initializesrwlock exclusive share depend thread want write access read access object protect lock win32 api support condition variable critical section srw lock repeatedly create delete thread expensive application service perform small amount work instantiation win32 thread pool provide user mode program service queue work request submit

submitthreadpoolwork function api

bind callback waitable handle regis- terwaitforsingleobject api work timer createthread- pooltimer waitforthreadpooltimercallbacks bind call- back o completion queue bindiocompletioncallback goal thread pool increase performance reduce memory footprint thread relatively expensive processor execute thread time matter thread available thread pool attempt reduce number runnable thread slightly delay work request reuse thread request provid- e thread effectively utilize machine cpu wait o- timer callback api allow thread pool reduce number thread process far few thread necessary process devote separate thread service waitable handle timer completion port fibe user mode code schedule accord user define scheduling algorithm fiber completely user mode facility kernel aware exist fiber mechanism use windows thread cpu execute fiber fiber cooperatively schedule mean preempt explicitly yield thread run fiber yield thread fiber schedule run time system programming language run time code system create fiber call convertthreadtofiber createfiber primary difference function createfiber begin execute fiber create begin execution application switchtofiber application terminate fiber call deletefiber fiber recommend thread use win32 api standard c library function potential incompatibility win32 user- mode thread thread environment block teb contain numerous thread field win32 api fiber share teb thread run lead problem win32 interface put state information teb fiber information overwrite different fiber fiber include win32 api facilitate porting legacy unix application write user mode thread model pthread user mode scheduling ums concrt new mechanism windows 7 user mode scheduling ums address limitation fiber note fiber unreliable execute win32 api teb thread run fiber block kernel user scheduler lose control cpu time kernel dispatcher take scheduling problem result fiber change kernel state thread priority impersonation token start asynchronous o. ums provide alternative model recognize windows thread actually thread kernel thread kt user thread ut type thread stack set save register kt ut appear single thread programmer uts block enter kernel implicit switch correspond kt

thread programmer uts block enter kernel implicit switch correspond kt take place ums use ut teb uniquely identify ut ut enter kernel explicit switch kt correspond ut identify current teb reason kernel know ut run uts invoke user mode scheduler fiber ums scheduler switch uts include switch ut enter kernel kt block happen kernel switch scheduling thread ums call primary thread use thread reenter user mode scheduler pick ut run eventually block kt complete operation ready return user mode ums reentere user mode scheduler run different ut ums queue ut correspond complete kt completion list user mode user mode scheduler choose new ut switch examine completion list treat ut list candidate scheduling key feature ums depict unlike fiber ums intend directly programmer detail write user mode scheduler challenging ums include scheduler scheduler come program- ming language library build ums microsoft visual studio 2010 ship concurrency runtime concrt concurrent programming framework c++ concrt provide user mode scheduler facility decompose program task schedule available cpu concrt provide support par style construct rudimentary resource management task synchronization primi- tive visual studio 2013 ums scheduling mode long available concrt significant performance metric show true parallel program write spend large time context-switch task benefit ums provide space outweigh complexity maintain separate scheduler case default nt scheduler perform well primary thread run user mode trap code switch park kt kt block primary return user mode kt unblock park queue ut completion ut completion list winsock windows socket api winsock session layer interface largely compatible bsd socket add windows extension provide standardized interface transport protocol different addressing scheme winsock application run winsock compliant protocol stack winsock undergo major update windows vista add tracing ipv6 support impersonation new security api feature winsock follow windows open system architecture wosa model provide standard service provider interface spi applica- tion

wosa model provide standard service provider interface spi applica- tion networking protocol application load unload layered protocol build additional functionality additional security transport protocol layer winsock support asynchronous opera- tion notification reliable multicasting secure socket kernel mode socket support simple usage model like wsaconnectbyname function accept target string specify ip address server service port number destination port ipc windows messaging win32 application handle interprocess communication way typical high performance way local rpc name pipe share kernel object name section object synchronization object event windows messaging facility approach particularly popular win32 gui application thread send message thread window call postmessage postthreadmessage sendmessage sendthreadmessage sendmessagecallback post message send message differ way post routine asynchronous return immediately call thread know message actually deliver send routine synchronous block caller message deliver process addition send message thread send datum mes- sage process separate address space datum copy system copy datum call sendmessage send message type wm copydata copydatastruct datum structure contain length address datum transfer message send windows copy datum new block memory give virtual address new block receive process win32 gui thread input queue receive message win32 application getmessage handle event input queue queue fill second task manager mark application respond note message pass- ing subject integrity level mechanism introduce early pro- cess send message wm copydata process high integrity level special windows api remove protection win32 api provide way application use memory virtual memory memory map file heap thread local storage awe physical application call virtualalloc reserve commit virtual memory virtualfree de commit release memory function enable application specify virtual address memory allocate random address select recommend security reason function operate multiple memory page size historical reason return memory allocate 64 kb boundary example function appear figure 21.11 virtu- alallocex virtualfreeex function allocate free memory separate process virtualallocexnuma leverage memory locality numa system

free memory separate process virtualallocexnuma leverage memory locality numa system way application use memory memory map file address space memory mapping convenient way process share memory process map file virtual memory memory mapping multistage process example figure 21.12 process want map address space share memory region process file need process call createfilemapping file handle 0xffffffff particular size optionally result file map object share inheritance lookup name handle duplication reserve 16 mb address space pvoid pbuf = virtualalloc(null 0x1000000 mem reserve | mem page readwrite commit upper 8 mb allocate space virtualalloc((lpvoid)((dword ptr)pbuf + 0x800000 0x800000 mem commit page readwrite memory decommit memory virtualfree((lpvoid)((dword ptr)pbuf + 0x800000 0x800000 release allocate address space virtualfree(pbuf 0 mem release code fragment allocate virtual memory heap provide way application use memory mal- loc free standard c new delete c++ heap win32 environment region pre committed address space win32 process initialize create default heap win32 set file mapping size 8 mb dword dwsize = 0x800000 open file create exist handle hfile = createfile(l"somefile.ext generic read | generic write file share read | file share write null open file attribute normal null create file mapping handle hmap = createfilemapping(hfile page readwrite | sec commit 0 dwsize l"shm 1 view space map pvoid pbuf = mapviewoffile(hmap file map access 0 0 0 dwsize map file unmap file code fragment memory mapping file application multithreade access heap synchronize protect heap space allocation data structure damage concurrent update multiple thread advantage heap allocation small 1 byte underlying memory page commit unfortunately heap memory share mark read heap allocation share page heapcreate programmer create heap mark read heapprotect create executable heap allocate specific numa node win32 provide heap management function process allocate manage private heap function heapcreate hea- palloc heaprealloc heapsize heapfree heapdestroy win32 api provide heaplock heapunlock function enable thread gain exclusive access heap note function perform synchronization truly lock page malicious buggy code bypass heap layer original win32 heap optimize efficient use space lead significant problem fragmentation address space large server program run long period time new low fragmentation heap lfh design introduce windows xp

greatly reduce fragmentation problem heap manager windows 7 later version automatically turn lfh appropriate additionally heap primary target attacker vulnerability double free use free memory corruption relate attack version windows include win- dows 10 add randomness entropy security mitigation prevent attacker guess ordering size location content fourth way application use memory thread local storage tls mechanism function rely global static datum typically fail work properly multithreaded environment instance c run time function strtok use static variable track current position parse string concurrent thread execute strtok cor- rectly need separate current position variable tls provide way maintain instance variable global function execute share thread tls provide dynamic static method create thread local storage dynamic method illustrate figure 21.13 tls mechanism allocate global heap storage attach thread environment block teb windows allocate user mode thread teb readily accessible thread tls thread state information user mode afinal way application use memory address windowing extension awe functionality mechanism allow developer directly reserve slot variable dword dwvarindex = t1salloc sure slot available dwvarindex = = tls index set value 10 value dword dwvar = dword)(dword ptr)t1sgetvalue(dwvarindex release index code dynamic thread local storage request free physical page ram memory manager allo- cateuserphysicalpages later commit virtual memory physical page virtualalloc request region phys- ical memory include scatter gather support user mode application access physical memory virtual address space useful 32 bit system 4 gb ram addition application bypass memory manager caching paging coloring algorithm similar ums awe offer way certain application extract additional performance customization windows offer default sql server example use awe memory use thread local static variable application declare variable follow ensure thread private copy declspec(thread dword cur pos = 0

microsoft design windows extensible portable operate sys- tem able advantage new technique hardware windows support multiple operating environment symmetric mul- tiprocessing include 32 bit 64 bit processor numa com- use kernel object provide basic service support client server computing enable windows support wide variety windows provide virtual memory integrated caching preemptive protect user datum guarantee program integrity windows support elaborate security mechanism exploit mitigation take advan- tage hardware virtualization windows run wide variety computer user choose upgrade hardware match budget performance requirement need alter application run include internationalization feature windows run variety country language windows sophisticated scheduling memory management algo- rithm performance scalability recent version windows add power management fast sleep wake feature decrease resource use area useful mobile system phone tablet windows volume manager ntfs file system provide sophisti- cat set feature desktop server system win32 api programming environment feature rich expansive allow programmer use windows feature program type operate system windows describe major list design goal windows describe detail describe booting process windows system describe main architectural layer windows kernel job object manager type service process manager provide local procedure responsibility o manager type networking windows support win- dow implement transport protocol describe networking pro- ntfs namespace organize ntfs handle data structure ntfs recover system crash guarantee recovery take place windows allocate user memory describe way application use memory win32 api russinovich et al 2017 deep overview windows 10 consider- able technical detail system internal component russinovich et al 2017 m. russinovich d. a. solomon a. ionescu win- dows internals 1

seventh edition microsoft press 2017 chapter 21 exercise circumstance use defer procedure call facility windows handle process obtain handle describe management scheme virtual memory manager vm manager improve performance describe useful application access page facility provide describe technique communicate datum local procedure setting conducive application different message passing technique manage caching windows cache manage ntfs directory structure differ directory struc- ture unix operate system process manage windows fiber abstraction provide windows differ thread abstraction user mode scheduling ums windows 7 differ fiber trade off fiber ums ums consider thread part ut kt useful allow ut continue execute parallel kt performance trade allow kts ut execute different processor self map occupy large amount virtual address space additional virtual memory self map easy vm manager page table page disk page table page keep windows system hibernate system power sup- pose change cpu ram hibernate system think work example show use suspend count helpful suspend resume thread windows modern operate system linux macos windows 10 influence early system discuss old highly influential operate system system xds-940 system kind system os/360 widely briefly cover old system long use provide comprehensive coverage additional system windows 7 freebsd mach windows 7 remain popular operating system user freebsd system unix system linux combine feature unix system freebsd base bsd model freebsd source code like linux source code freely available mach provide compatibility bsd unix especially interesting bsd mach form architecture ios macos popular modern understand fundamental concept operating system cpu scheduling memory management process position examine concept apply old highly influential operating system xds-940 system kind system os/360 widely order presentation highlight similarity difference system strictly chronological order importance student operating system familiar system bibliographical note end chapter include reference read early system paper write design- er system important technical content style flavor explain operating system feature

system important technical content style flavor explain operating system feature migrate time large com- puter system small one discuss feature historically important operating system reason study early architecture operating system feature run huge system eventually way small system examination operate system mainframe microcomputer show feature available main- frame adopt microcomputer operating system concept appropriate class computer mainframe minicomputer microcomputer handheld understand modern oper- ating system need recognize theme feature migration long history operating system feature show figure a.1 agood example feature migration start multiplexed informa- tion computing services multics operate system multics devel- influentia operating systems migration operate system concept feature ope 1965 1970 massachusetts institute technology mit compute utility run large complex mainframe computer ge-645 idea develop multics subsequently bell laboratories original partner development multics design unix unix operate system design 1970 pdp-11 minicomputer 1980 feature unix basis unix like operate system microcomputer feature include recent operating system microcomputer microsoft windows windows xp macos operating system linux include feature find pda turn attention historical overview early computer system note history computing start far computer loom calculator begin discussion com- puter twentieth century 1940s compute device design implement perform specific fix task modify task require great deal effort manual labor change 1940 alan turing john von neumann colleague separately work idea general purpose store program computer machine program store data store program store provide instruction datum fundamental computer concept quickly generate number general purpose computer history machine blur time secrecy development world war ii likely working store program general purpose computer manchester mark 1 run successfully 1949 commercial computer ferranti mark 1 go sale 1951 early computer physically enormous machine run console programmer operator computer system write program operate directly operator console program load manually memory panel switch instruction time paper tape punch card appropriate button push set start address

paper tape punch card appropriate button push set start address start execution program program run program- mer operator monitor execution display light console error discover programmer halt program examine content memory register debug program directly console output print punch paper tape card dedicated computer systems time go additional software hardware develop card reader line printer magnetic tape commonplace assembler loader linker design ease programming task library common function create common function copy new program have write provide software routine perform o especially important new o device characteristic require careful programming special subroutine call device driver write o device device driver know buffer flag register control bit status bit particular device type device driver simple task read character paper tape reader involve complex sequence device specific operation write necessary code time device driver simply later compiler fortran cobol language appear mak- e programming task easy operation computer complex prepare fortran program execution example pro- grammer need load fortran compiler computer compiler normally keep magnetic tape proper tape need mount tape drive program read card reader write tape fortran compiler produce assembly language output assemble procedure require mount tape assembler output assem- bler need link support library routine finally binary object form program ready execute load memory debug console influentia operating systems significant setup time involve running job job consist separate step 1 load fortran compiler tape 2 run compiler 3 unload compiler tape 4 load assembler tape 5 run assembler 6 unload assembler tape 7 load object program 8 run object program error occur step programmer operator start beginning job step involve loading unloading magnetic tape paper tape punch card job setup time real problem tape mount programmer operate console cpu sit idle remember early day computer available expensive computer cost million dollar include operational cost power cooling programmer computer time

million dollar include operational cost power cooling programmer computer time extremely valuable owner want computer possible need high utilization shared computer systems solution twofold professional computer operator hire programmer long operate machine soon job finish operator start operator experi- ence mount tape programmer setup time reduce programmer provide card tape need short description job run course operator debug incorrect program console operator understand program case program error dump memory register take programmer debug dump dump memory register allow operator continue immediately job leave programmer difficult second job similar need batch run computer group reduce setup time instance suppose operator receive fortran job cobol job fortran job run order set fortran load compiler tape set cobol set fortran run fortran program batch setup fortran save operator time memory layout resident monitor problem example job stop operator notice stop observe console determine stop normal abnormal termination dump memory register necessary load appropriate device job restart computer transition job cpu overcome idle time people develop automatic job sequencing technique rudimentary operate system create small program call resident monitor create transfer control automatically job figure a.2 resident monitor memory resident computer turn resident monitor invoke transfer control program program terminate return control resident monitor program resident monitor automatically sequence program job resident monitor know program execute previously operator give short description program run datum control card introduce provide information directly monitor idea simple addition program datum job programmer supply control card contain directive resident monitor indicate program run example normal user program require program run fortran compiler ftn assembler asm user program run use separate control card $ ftn execute fortran compiler $ asm execute assembler $ run execute user program card tell resident monitor program run influentia operating systems use additional control card define boundary $ job card job $ end final

card define boundary $ job card job $ end final card job card useful account machine resource programmer parameter define job account number charge control card define function ask operator load unload tape problem control card distinguish datum program card usual solution identify special character pattern card system dollar sign character $ control language jcl slash mark column figure a.3

show sample

card deck setup simple batch system resident monitor identifiable part control card interpreter responsible read carry instruction card point execution loader invoke control card interpreter load system pro- grams application program memory interval device driver control card interpreter loader system o device system application program link device driver provide continuity operation save memory space programming time batch system work fairly resident monitor provide auto- matic job sequencing indicate control card control card indicate program run monitor load program mem- ory transfer control program complete transfer control datum program program compile card deck simple batch system monitor read control card load appropriate program cycle repeat control card interpret job monitor automatically continue job switch batch system automatic job sequencing improve performance problem simply human consid- erably slow computer consequently desirable replace human operation operating system software automatic job sequencing elimi- nate need human setup time job sequencing arrangement cpu idle problem speed mechanical o device intrinsically slow electronic device slow cpu work microsecond range thousand instruction execute second fast card reader contrast read 1,200 card minute 20 card second difference speed cpu o device order magnitude time course improvement technology result fast o device unfortunately cpu speed increase fast problem unresolved exacerbate common solution o problem replace slow card reader input device line printer output device magnetic tape unit computer system late 1950 early 1960 batch system read card reader write line printer card punch cpu read directly card instead card copy magnetic tape separate device tape sufficiently take carry computer card need input program equivalent record read tape similarly output write tape content tape print later card reader line printer operate line main computer figure a.4 obvious advantage line operation main computer long constrain speed card reader line printer limit speed fast magnetic tape unit technique magnetic tape o apply

speed fast magnetic tape unit technique magnetic tape o apply operation o device line b line influentia operating systems similar equipment card reader card punch plotter paper tape real gain line operation come possibility multiple reader tape tape printer system cpu cpu process input twice fast reader read card reader work simultaneously produce tape cpu busy disadvantage long delay get particular job run job read tape wait additional job read tape fill tape rewind unload hand carry cpu mount free tape drive process unreasonable batch system course similar job batch tape take computer line preparation job continue time quickly replace system disk system widely available greatly improve line operation problem tape system card reader write end tape cpu read entire tape write rewind read tape nature sequential access device disk system eliminate problem random access device head move area disk switch rapidly area disk card reader store new card position need cpu read card disk system card read directly card reader disk location card image record table keep operate system job execute operate system satisfy request card- reader input read disk similarly job request printer output line line copy system buffer write disk job complete output actually print form processing call spooling figure a.5 acronym simultaneous peripheral operation line spooling essence use disk huge buffer read far ahead possible input device store output file output device able accept spooling process datum remote site cpu send datum communication path remote printer accept entire input job remote card reader remote processing speed cpu intervention cpu need notify processing complete spool batch datum spooling overlap o job computation job simple system spooler read input job print output different job time job job execute read card disk print output line disk spooling direct beneficial effect performance system cost disk space table computation job o

effect performance system cost disk space table computation job o job place time spooling cpu o device work high rate spooling lead naturally multiprogramming foundation modern atlas operate system design university manchester england late 1950 early 1960 basic feature novel time standard part modern operate system device driver major system addition system call add set special instruction call extra code atlas batch operate system spooling spooling allow system schedule job accord availability peripheral device magnetic tape unit paper tape reader paper tape punch line printer card reader card punch remarkable feature atlas memory manage- ment core memory new expensive time computer drum main memory small core memory cache drum demand paging transfer information core memory drum automatically atlas system british computer 48 bit word address 24 bit encode decimal allow 1 million word address time extremely large address space physical memory atlas 98 kb word drum 16 kb word core memory divide 512 word page provide 32 frame physical memory associative memory 32 register implement mapping virtual address physical address page fault occur page replacement algorithm invoke memory frame keep drum transfer start immediately page replacement algorithm attempt predict future memory access behavior base past behavior reference bit frame set frame access reference bit read memory 1,024 instruction 32 value bit retain history define time recent ref- influentia operating systems erence t1 interval reference t2 page choose replacement following order 1 page t1 > t2 + 1 consider long use 2 t1 t2 page replace page large value t2 page replacement algorithm assume program access memory loop time reference t2 reference expect t2 time unit later reference occur t1 > t2 assume page long page replace page use page need long time replace time reference expect t2 t1 xds-940 operate system design university california berkeley early 1960s like atlas system

system design university california berkeley early 1960s like atlas system paging memory management unlike atlas system time share system paging relocation demand paging virtual memory user process 16 kb word physical memory 64 kb word page 2 kb word page table keep register physical memory large virtual memory user process memory time number user increase page sharing page contain read reentrant code process keep drum swap memory necessary xds-940 system construct modify xds-930 mod- ification typical change basic computer allow operate system write properly user monitor mode add certain instruction o halt define privileged attempt execute privileged instruction user mode trap system instruction add user mode instruction set instruction create new resource file allow operat- e system manage physical resource file example allocate 256 word block drum bitmap manage free drum block file index block pointer actual datum block index block chain xds-940 system provide system call allow process cre- eat start suspend destroy subprocesse programmer construct system process separate process share memory communi- cation synchronization process creation define tree structure process root subprocesse node tree subprocesse turn create subprocesse operate system design technische hogeschool eindhoven netherlands mid-1960 batch system run dutch computer el x8 32 kb 27 bit word system mainly note clean design particularly layer structure use set concurrent process employ semaphore synchronization unlike process xds-940 system set process system static operate system design set cooperate process addition user process create serve active agent compile execute print user program job finish process return input queue select priority cpu scheduling algorithm priority recom- put 2 second inversely proportional cpu time recently 8 10 second scheme give high priority o bind process new process memory management limit lack hardware support how- system limit user program write algol software paging scheme algol compiler automatically generate call system routine sure request information

algol compiler automatically generate call system routine sure request information memory swap necessary back store 512 kb word drum 512 word page lru page replacement strategy major concern system deadlock control banker algorithm provide deadlock avoidance closely relate system venus system venus system layer structured design semaphore synchronize process low level design implement microcode pro- vide fast system page segment memory memory management addition system design time share system batch system rc 4000 system like system notable primarily design concept design late 1960 danish 4000 computer regnecentralen particularly brinch hansen objective design batch system time share system specific system goal create operate system nucleus kernel complete operate system build system structure layered low level comprise kernel provide kernel support collection concurrent process round robin cpu scheduler process share memory primary communication synchronization mechanism message system pro- vide kernel process communicate exchang-e fixed sized message word length message store buffer common buffer pool message buffer long require return common pool influentia operating systems message queue associate process contain message send process receive message remove queue fifo order system support primitive operation execute atomically send message receiver message buffer wait message sender message buffer send answer result message buffer wait answer result message buffer operation allow process exchange message primitive require process service message queue fifo order block process handle message remove restriction developer provide additional commu- nication primitive allow process wait arrival message answer service queue order wait event previous buffer buffer result event buffer o device treat process device driver code convert device interrupt register message pro- cess write terminal send terminal message device driver receive message output character terminal input character interrupt system transfer device driver device driver create message input character send compatible time sharing system ctss design mit exper- imental time sharing system appear 1961 implement

design mit exper- imental time sharing system appear 1961 implement provide set interactive command allow manip- ulate file compile run program terminal 7090 32 kb memory 36 bit word monitor 5 kb word leave 27 kb user user memory image swap memory fast drum cpu scheduling employ multilevel- feedback queue algorithm time quantum level 2 i time unit program finish cpu burst time quantum move level queue give twice time program high level short quantum run initial level program determine size time quantum long swap time ctss extremely successful use late 1972 limit succeed demonstrate time sharing con- venient practical mode computing result ctss increase development time sharing system result development multics operate system design 1965 1970 mit natural extension ctss ctss early time sharing system successful create immediate desire proceed quickly big well system large computer available designer ctss set create time sharing utility compute service provide like electrical power large computer system connect telephone wire terminal office home city operate system time share system run continuously vast file system share program datum multics design team mit ge later sell com- puter department honeywell bell laboratories drop project 1969 basic ge 635 computer modify new com- puter system call ge 645 mainly addition page segmentation multics virtual address compose 18 bit segment number 16 bit word offset segment page 1 kb word page second chance page replacement algorithm segmented virtual address space merge file system segment file segment address file file system multilevel tree structure allow user create like ctss multics multilevel feedback queue cpu scheduling protection accomplish access list associate file set protection ring execute process system write entirely pl/1 comprise 300,000 line code extend multiprocessor system allow cpu take service maintenance system continued run prime example development common o subroutine follow development resident monitor privileged instruction memory

common o subroutine follow development resident monitor privileged instruction memory protec- tion simple batch processing system develop separately computer different language different system software ning complete range small business machine large scientific machine set software need system operate system os/360 arrangement intend influentia operating systems unfortunately os/360 try thing people result task especially file system include type field define type file different file type define fix length variable length record blocked unblocked file contiguous allocation user guess size output file job control language jcl add parameter possible option make incomprehensible average user memory management routine hamper architecture base register addressing mode program access modify base register absolute address generate cpu arrangement prevent dynamic relocation program bind physical memory load time separate version operate system produce os mft fixed region os mvt variable region system write assembly language thousand program- mer result million line code operate system require large amount memory code table operate system overhead consume half total cpu cycle year new version release add new feature fix error fix error cause remote system number know error system remain fairly constant architecture underlie hardware provide segment page virtual memory new version os hardware different way os vs1 create large virtual address space run os mft virtual memory operate system page user program os vs2 release 1 run os mvt virtual memory finally os vs2 release 2 call mvs provide user virtual memory mvs basically batch operate system ctss system run multics tss/360 suppose large time share utility basic 360 architecture modify model 67 provide virtual memory site purchase 360/67 anticipation tss/360 tss/360 delay time share system devel- ope temporary system tss/360 available time sharing option single user system cp/67 provide virtual machine run tss/360 eventually deliver failure large slow result site switch temporary system mvs cms cp/67 rename vm tss/360

site switch temporary system mvs cms cp/67 rename vm tss/360 multics achieve commercial success go wrong problem advanced system large complex understand problem assumption computing power available large remote source minicom- cp m ms dos puter come decrease need large monolithic system follow workstation personal computer com- put power close close end user dec create influential computer system history probably famous operating system associate dec vms popular business orient system use today openvms product hewlett packard influential dec operate system tops-20 start life research project bolt beranek newman bbn 1970 bbn take business orient dec pdp-10 computer run- ning tops-10 add hardware memory page system implement virtual memory write new operate system computer advan- tage new hardware feature result tenex general purpose time share system dec purchase right tenex create new computer build hardware pager result system decsystem-20 tops-20 operate system tops-20 advanced command line interpreter provide help need user combination power computer reasonable price decsystem-20 popular time sharing system time 1984 dec stop work line 36 bit pdp-10 computer concentrate 32 bit vax system run vms cp m ms dos early hobbyist computer typically build kit run single pro- gram time system evolve advanced system computer component improve early standard operating system com- puter 1970 cp m short control program monitor write gary kindall digital research inc.

cp m run primarily personal computer cpu 8 bit intel 8080 cp m originally support 64 kb memory run program time course text base command interpreter command interpreter resemble operate system time tops-10 dec gates company write new operate system 16 bit cpu choice intel 8086 operating system ms dos similar cp m rich set build command model tops-10 ms dos popular personal computer operating system time start 1981 continue development 2000 support 640 kb memory ability address extended expand memory somewhat limit lack fundamental current operating system feature especially protect memory influentia operating systems a.12 macintosh operating system windows advent 16 bit cpu operate system personal computer advanced feature rich usable apple macintosh computer arguably computer gui design home user certainly successful start launch 1984 mouse screen pointing selecting come utility program take advantage new user interface hard disk drive relatively expensive 1984 come 400 kb capacity floppy drive default original mac os run apple computer slowly eclipse microsoft windows start version 1.0 1985 license run different computer multitude company micro- processor cpu evolve 32 bit chip advanced feature pro- tected memory context switching operate system add feature previously find mainframe minicomputer time personal computer powerful system use- ful purpose minicomputer die replace general- special purpose server personal computer continue increase capacity performance server tend stay ahead memory disk space number speed available cpu today server typically run data center machine room personal computer sit desk talk server network desktop rivalry apple microsoft continue today new version windows mac os try outdo fea- ture usability application functionality operating system amigaos os/2 appear time long term competitor lead desktop operating system linux form continue gain popularity technical user nontechnical user system like laptop child olpc child connect computer network http://laptop.org/ mach operate system trace ancestry accent operate sys- tem develop

mach operate system trace ancestry accent operate sys- tem develop carnegie mellon university cmu mach communication system philosophy derive accent significant portion system example virtual memory system task thread management develop scratch work mach begin mid 1980 operate system design following critical goal mind 1 emulate 4.3 bsd unix executable file unix system run correctly mach 2

modern operate system support memory model parallel distributed computing 3 kernel simple easy modify 4.3 bsd mach development follow evolutionary path

bsd unix sys- tem mach code initially develop inside 4.2 bsd kernel bsd ker- nel component replace mach component mach component complete bsd component update 4.3 bsd available 1986 virtual memory communication subsystem run dec vax computer family include multiprocessor version shortly 1987 see completion encore multimax sequent balance multiprocessor version include task thread support official release system release 0 release 1 release 2 mach provide compatibility correspond bsd system include bsd code kernel new feature capability mach kernel release large cor- respond bsd kernel mach 3 move bsd code outside kernel leave small microkernel system implement basic mach fea- ture kernel unix specific code evict run user mode server exclude unix specific code kernel allow replacement bsd operate system simultaneous execution multi- ple operating system interface microkernel addition bsd user mode implementation develop dos macintosh oper- ating system osf/1 approach similarity virtual machine concept virtual machine define software mach kernel interface hardware release 3.0 mach available dec machine multiprocessor dec sequent encore system mach propel forefront industry attention open software foundation osf announce 1989 use mach 2.5 basis new operate system osf/1 mach 2.5 basis operate system workstation brainchild steve jobs apple computer fame initial release osf/1 occur year later system compete unix system v release 4 operate system choice time unix international ui member osf member change direction dec unix base mach kernel unlike unix develop regard multiprocessing mach incorporate multiprocessing support support exceedingly flexible range share memory system system memory share processor mach use lightweight process form multiple thread execution task address space support multiprocessing parallel computation extensive use mes- sage communication method ensure protection mechanism complete efficient integrate message virtual memory system mach ensure message handle efficiently finally have virtual memory system use message communicate dae- mon manage backing store mach provide great flexibility design implementation

mon manage backing store mach provide great flexibility design implementation memory object manage task provide low level primitive system call complex function build mach reduce size kernel permit operating system influentia operating systems today remain pure mach implementation gnu hurd little operate system mach live xnu kernel drive macosand iosvariant xnu codebase apple obtain acquisition nextstep operate system mach core layer bsd api apple continue support maintain mach api accessible specialized system call know trap mach messages kernel continues evolve new feature day previous edition operating system concepts include entire chapter mach chapter appear fourth edition available web http://www.os-book.com a.14 capability base system hydra cap section survey capability base protection system sys- tem differ complexity type policy imple- mente system widely provide interesting proving ground protection theory hydra capability base protection system provide considerable flex- ibility system implement fixed set possible access right include basic form access right read write execute memory seg- ment addition user protection system declare right interpretation user define right perform solely user program system provide access protection use right use system define right facility constitute significant development protection technology operation object define procedurally procedure imple- ment operation form object access indirectly capability name user define procedure iden- tifie protection system deal object user define type definition object know hydra name operation type auxiliary right auxiliary right describe capability instance type process perform operation type object capability hold object con- tain operation invoke auxiliary right restriction enable discrimination access right instance- instance process process basis hydra provide right amplificatio scheme allow procedure certify trustworthy act formal parameter specify type behalf process hold right execute procedure right hold trustworthy procedure independent exceed right hold call process procedure regard universally trustworthy procedure allow act type instance trustworthiness extend procedure program segment execute process capability base systems hydra cap

procedure program segment execute process capability base systems hydra cap amplification allow implementation procedure access representa- tion variable abstract data type process hold capability type object instance capability include auxiliary right invoke operation p include call kernel right read write execute segment represent a. capability give process means indirect access operation p representation specific purpose process invoke operation p object capability access amplify control pass code body p. amplification necessary allow p right access storage segment represent implement operation p define abstract data type code body p allow read write segment directly call process return p capability restore original unamplified state case typical right hold process access protect segment change dynamically depend task perform dynamic adjustment right perform guarantee consistency programmer define abstraction amplification right state explicitly declaration abstract type hydra operating user pass object argument procedure need ensure procedure modify object implement restriction readily pass access right modifi- cation write right amplification occur right modify reinstate user protection requirement circumvent general course user trust procedure perform task cor- rectly assumption correct hardware software error hydra solve problem restrict amplification procedure mechanism hydra design direct solution problem mutually suspicious subsystem problem define follow suppose program invoke service number different user example sort routine compiler game user invoke service program risk program malfunc- tion damage give datum retain access right datum authority later similarly service program private file accounting purpose example access directly call user program hydra provide mechanism directly deal problem hydra subsystem build protection kernel require protection component subsystem interact kernel call set kernel define primitive define access right resource define subsystem subsystem designer define poli- cie use resource user process policy enforce use standard access protection provide capability system programmer direct use protection system acquaint- e feature appropriate reference manual hydra provide large library system define procedure call

reference manual hydra provide large library system define procedure call user program programmer explicitly incorporate call system procedure program code use program translator interface hydra influentia operating systems cambridge cap system different approach capability base protection take design cambridge cap system cap capability system simple superficially powerful hydra close examination show provide secure protection user define object cap kind capability ordinary kind call data capability provide access object right pro- vide standard read write execute individual storage seg- ment associate object data capability interpret microcode cap machine second kind capability call software capability protect interpret cap microcode interpret pro- tected privileged procedure write application programmer subsystem particular kind right amplification associate protect procedure execute code body procedure process temporarily acquire right read write content software capability specific kind right amplifica- tion correspond implementation seal unseal primitive capability course privilege subject type verification ensure software capability specify abstract type pass procedure universal trust place code cap machine microcode bibliographical note end chapter interpretation software capability leave completely sub- system protect procedure contain scheme allow variety protection policy implement programmer define protect procedure incorrect security overall system compromise basic protection system allow unverified user define protect procedure access storage segment capability belong protection environment reside consequence insecure protect procedure protection breakdown subsystem procedure responsibility designer cap system note use software capability allow realize considerable economy formulate implement protection policy commensurate requirement abstract resource subsystem designer want use facility simply study reference manual case hydra instead learn principle technique protection system provide library procedure a.15 system course operate system interest- e property mcp operate system burroughs computer family write system program language support segmentation multiple cpu scope operate system cdc 6600 multi cpu system coordination synchronization multiple process surprisingly design history litter operating system suit purpose time long short time fade replace operate

system suit purpose time long short time fade replace operate system feature support new hardware easy use well market sure trend continue future loom calculator describe frah 2001 show graphically manchester mark 1 discuss rojas hashagen 2000 offspring ferranti mark 1 describe ceruzzi 1998 kilburn et al 1961 howarth et al 1961 examine atlas oper- xds-940 operate system describe lichtenberger pirtle operating system cover dijkstra 1968 mckeag wilson 1976 venus system describe liskov 1972 brinch hansen 1970 brinch hansen 1973 discuss rc 4000 compatible time sharing system

ctss present corbato et al multics operate system describe corbato vyssotsky 1965 organick 1972 cp/67 describe meyer seawright 1970 parmelee et al dec vms discuss kenah et al 1988 tenex describe bobrow et al 1972 description apple macintosh appear apple 1987 information operate system history freiberger mach operate system ancestor accent operate system describe rashid robertson 1981 mach communication system cover rashid 1986 tevanian et al 1989 accetta et al 1986 mach scheduler describe detail tevanian et al 1987a black 1990 early version mach shared- memory memory map system present tevanian et al 1987b good resource describe mach project find mckeag wilson 1976 discuss mcp operate system burroughs computer family scope operate system cdc hydra system describe wulf et al 1981 cap system describe needham walker 1977 organick 1972 discuss multics ring protection system influentia operating systems accetta et al 1986 m. accetta r. baron w. bolosky d. b. golub r. rashid a. tevanian m. young mach new kernel foundation unix devel- opment

proceedings summer usenix conference 1986 page 93–112 apple technical introduction macintosh family d. l. black scheduling support concurrency paral- lelism mach operating system ieee computer volume 23 number 5 1990 page 35–43 bobrow et al 1972 d. g. bobrow j. d. burchfiel d. l. murphy r. s. tom- linson tenex paged time sharing system pdp-10 communications acm volume 15 number 3 1972 p. brinch hansen nucleus multiprogram- ming system communications acm volume 13 number 4 1970 page 238–241 250 p. brinch hansen operating system principles prentice p. e. ceruzzi history modern computing mit press 1998 corbato vyssotsky 1965 f. j. corbato v. a. vyssotsky introduction overview multics system proceedings afips fall joint computer conference 1965 page 185–196 corbato et al 1962 f. j. corbato m. merwin daggett r. c. daley experimental time sharing system proceedings

afips fall joint computer conference 1962 page 335–344 e. w. dijkstra structure multiprogramming system communications acm volume 11 number 5 1968 page 341 g. frah universal history computing john wiley sons m. frauenfelder computer illustrated history carlton books 2005 freiberger swaine 2000 p. freiberger m. swaine fire valley making personal computer mcgraw hill 2000 howarth et al 1961 d. j. howarth r. b. payne f. h. sumner manchester university atlas operating system ii user description computer journal volume 4 number 3 1961 page 226–229 kenah et al 1988 l. j. kenah r. e. goldenberg s. f. bate vax vms internals data structures digital press 1988 kilburn et al 1961 t. kilburn d. j. howarth r. b. payne f. h. sumner manchester university atlas operating system internal organiza- tion computer journal volume 4 number 3 1961 page 222–225 lett konigsford 1968 a. l. lett w. l. konigsford tss/360 time share operating system proceedings afips fall joint computer conference 1968 page 15–28 lichtenberger pirtle 1965 w. w. lichtenberger m. w. pirtle facility experimentation man machine interaction proceedings afips fall joint computer conference 1965 page 589–598 b. h. liskov design venus operating system com- munications acm volume 15 number 3 1972 page 144–149 mckeag wilson 1976 r. m. mckeag r. wilson studies operating systems academic press 1976 mealy et al 1966 g. h. mealy b. i. witt w. a. clark functional meyer seawright 1970 r.

a. meyer l. h. seawright virtual 1970 page 199–218 needham walker 1977 r. m. needham r. d. h. walker cam- bridge cap computer protection system proceedings sixth sym- posium operating system principles 1977 page 1–10 e. i. organick multics system examination struc- ture mit press 1972 parmelee et al 1972 r. p. parmelee t. i. peterson c. c. tillman d. hat- volume 11 number 2 1972 page 99–130

r. f. rashid rig accent mach evolution network operating system proceedings acm ieee computer society fall joint computer conference 1986 page 1128–1137 rashid robertson 1981 r. rashid g. robertson accent com- munication orient network operating system kernel proceedings acm symposium operating system principles 1981 page 64–75 rojas hashagen 2000 r. rojas u. hashagen computers history architectures mit press 2000 tevanian et al 1987a a. tevanian jr. r. f. rashid d. b. golub d. l. black e. cooper m. w. young mach thread unix kernel battle control proceedings summer usenix conference 1987 tevanian et al 1987b a. tevanian jr. r. f. rashid m. w. young d. b. golub m. r. thompson w. bolosky r. sanzi unix interface shared memory memory map files mach technical report carnegie- mellon university 1987 tevanian et al 1989 a. tevanian jr. b. smith mach model future unix byte 1989 influentia operating systems wulf et al 1981 w. a. wulf r. levin s. p. harbison hydra c.mmp experimental computer system mcgraw hill 1981 update

dave probert microsoft windows 7 operate system 32-/64 bit preemptive mul- titasking client operating system microprocessor implement intel ia-32 amd64 instruction set architecture isas microsoft corresponding server operate system windows server 2008 r2 base code windows 7 support 64 bit amd64 ia64 itanium isa windows 7 late series microsoft operate system base nt code replace early system base windows 95/98 appendix discuss key goal windows 7 layered architecture system easy use file system networking feature programming interface explore principle underlie windows 7 design specific component system provide detailed discussion windows 7 file system illustrate networking protocol support windows 7 describe interface available windows 7 system application describe important algorithm implement windows 7 system write assembly language single processor intel develop new technology nt portable operating system support os/2 posix application programming interface apis october 1988 dave cutler architect dec vax vms operate system hire give charter build microsoft new operating system originally team plan use os/2 api nt native environ- ment development nt change use new 32 bit windows api call win32 base popular 16 bit api windows 3.0 version nt windows nt 3.1 windows nt 3.1 advanced server time 16 bit windows version 3.1 windows nt ver- sion 4.0 adopt windows 95 user interface incorporate internet web server web browser software addition user interface routine graphic code move kernel improve performance effect decrease system reliability previous version nt port microprocessor architecture windows 2000 version release february 2000 support intel compatible processor marketplace factor windows 2000 incorporate signifi- nt change add active directory x.500 base directory service well networking laptop support support plug play device distribute file system support processor memory october 2001 windows xp release update win- dows 2000 desktop operate system replacement windows 95/98 2002 server edition windows xp available call windows .net server windows xp update graphical

xp available call windows .net server windows xp update graphical user interface gui visual design take advantage recent hardware advance new ease use feature numerous feature add automat- ically repair problem application operate system result change windows xp provide well networking device experience include zero configuration wireless instant messaging stream- e medium digital photography video dramatic performance improve- ment desktop large multiprocessor well reliability security early windows operate system long await update windows xp call windows vista release november 2006 receive win- dow vista include improvement later show windows 7 improvement overshadow windows vista perceive sluggishness compatibility problem microsoft respond criticism windows vista improve engineering process work closely maker windows hardware application result windows 7 release october 2009 correspond server edition windows significant engineering change increase use execution tracing counter profiling analyze system behavior tracing run constantly system watch hundred scenario execute scenario fail succeed perform trace analyze determine cause windows 7 use client server architecture like mach implement operating system personality win32 posix user level process call subsystem time windows support os/2 subsystem remove windows xp demise os/2 subsystem architecture allow enhancement operating system person- ality affect application compatibility posix subsystem continue available windows 7 win32 api popular posix api site sub- system approach continue interesting study operating system perspective machine virtualization technology dominant way run multiple operate system single machine windows 7 multiuser operating system support simultaneous access distribute service multiple instance gui windows terminal service server edition windows 7 support simultaneous terminal server session windows desktop system desktop edition terminal server multiplex keyboard mouse mon- itor virtual terminal session log user feature call fast user switching allow user preempt console pc have log log note early gui implementation move kernel mode windows nt 4.0 start user mode windows vista include desktop window manager dwm user mode process

windows vista include desktop window manager dwm user mode process dwm implement desktop compositing windows provide windows aero interface look windows directx graphic software directx continue run kernel code implement win- dows previous windowing graphic model win32k gdi windows 7 substantial change dwm significantly reduce memory footprint improve performance windows xp version windows ship 64 bit version ia64 2001 amd64 2005 internally native nt file system ntfs win32 api 64 bit integer appropriate major extension 64 bit windows xp support large virtual address 64 bit edition windows support large physical memory time windows 7 ship amd64 isahad available cpu intel amd addi- tion time physical memory client system frequently exceed 4 gb limit ia-32 result 64 bit version windows 7 commonly instal large client system amd64 architecture support high fidelity ia-32 compatibility level individual process 32- 64 bit application freely mix single system rest description windows 7

distinguish client edition windows 7 correspond server edition base core component run binary file kernel driver similarly microsoft ship variety different edition release address different market price point difference edition reflect core system chapter focus primarily core component windows 7 microsoft design goal windows include security reliability windows posix application compatibility high performance extensibility portabil- ity international support additional goal energy efficiency dynamic device support recently add list discuss goal achieve windows 7 windows 7 security goal require adherence design stan- dard enable windows nt 4.0 receive c2 security classification u.s. government c2 classification signify moderate level protection defective software malicious attack classification define department defense trusted computer system evaluation criteria know orange book extensive code review testing combine sophisticated automatic analysis tool identify investigate potential defect represent security vulnerability windows base security discretionary access control system object include file registry setting kernel object protect access- control list acls section 13.4.2 acl vulnerable user pro- grammer error common attack consumer system user trick run code browse web windows 7 include mechanism call integrity level act rudimentary capability system control access object process mark have low medium high integrity windows allow process modify object high integrity level matter setting acl security measure include address space layout randomization aslr nonexecutable stack heap encryption digital signature facility aslr thwart form attack prevent small amount inject code jump easily code load process normal operation safeguard make likely system attack fail crash let attack code control recent chip intel amd base amd64 architec- ture allow memory page mark contain executable instruction code windows try mark stack memory heap execute code prevent attack program bug allow buffer overflow trick execute content buffer technique apply program rely modify datum execute column label data execution prevention windows task manager show process mark prevent attack windows use encryption common

manager show process mark prevent attack windows use encryption common protocol communicate securely website encryption protect user file store disk pry eye windows 7 allow user easily encrypt virtually disk removable storage device usb flash drive feature call bitlocker computer encrypt disk steal thief need sophisticated technology electron microscope gain access computer file windows use digital signature sign operating system binary verify file produce microsoft know company edition windows code integrity module activate boot ensure load module kernel valid signature assure tamper line attack windows mature greatly operate system year lead windows 2000 time reliability increase factor maturity source code extensive stress testing system improved cpu architecture automatic detection error driver microsoft party windows subsequently extend tool achieve reliability include automatic analysis source code error test include provide invalid unexpected input parameter know fuzzing detect validation failure application version driver verifier apply dynamic checking extensive set com- mon user mode programming error improvement reliability result move code kernel user mode service windows provide extensive support write driver user mode system facility kernel user mode include desktop window manager software stack audio significant improvement windows experience come add memory diagnostic option boot time addition especially valuable consumer pc error correct mem- ory bad ram start drop bit result frustratingly erratic behavior system availability memory diagnostic greatly reduce stress level user bad ram windows 7 introduce fault tolerant memory heap heap learn application crash automatically insert mitigation future execution application crash make application reliable contain common bug memory free access past end allocation achieve high reliability windows particularly challenging billion computer run windows reliability problem affect small percentage user impact tremendous number human being complexity windows ecosystem add challenge million instance application driver software constantly download run windows system course constant stream malware attack windows hard attack directly exploit increasingly target popular

malware attack windows hard attack directly exploit increasingly target popular application cope challenge microsoft increasingly rely commu- nication customer machine collect large amount datum ecosystem machine sample perform software run problem encounter cus- tomer send datum microsoft system software crash hang constant stream datum customer machine collect care- fully user consent invade privacy result microsoft build improve picture happen windows ecosystem allow continuous improvement software update provide datum guide future release windows windows posix application compatibility mention windows xp update windows 2000 replacement windows 95/98 windows 2000 focus primarily compat- ibility business application requirement windows xp include great compatibility consumer application run win- dows 95/98 application compatibility difficult achieve application check particular version windows depend extent quirk implementation api latent application bug mask previous system forth application compile different instruction set windows 7 implement strategy run application despite incompatibility like windows xp windows 7 compatibility layer sit application win32 api layer make windows 7 look bug bug compatible previous version windows windows 7 like early nt release maintain support run 16 bit application thunking conversion layer translate 16 bit api call equivalent 32 bit call similarly 64 bit version windows 7 provide thunking layer translate 32 bit api call native 64 bit call windows subsystem model allow multiple operate system person- alitie support note early api commonly windows win32 api edition windows 7 support posix subsystem posix standard specification unix allow available unix compatible software compile run modification final compatibility measure edition windows 7 provide virtual machine run windows xp inside windows 7 allow applica- tion bug bug compatibility windows xp windows design provide high performance desktop system largely constrain o performance server system cpu bottleneck large multithreaded multiprocessor envi- ronment locking performance cache line management key scalability satisfy performance requirement nt variety tech- nique asynchronous o optimize protocol network

requirement nt variety tech- nique asynchronous o optimize protocol network kernel- base graphic rendering sophisticated caching file system datum memory management synchronization algorithm design awareness performance consideration relate cache line multi- windows nt design symmetrical multiprocessing smp multiprocessor computer thread run time kernel cpu windows nt use priority base preemptive scheduling thread execute kernel dispatcher interrupt level thread process run windows preempt higher- priority thread system respond quickly chapter 5 subsystem constitute windows nt communicate efficiently local procedure lpc facility provide high performance message passing thread request synchronous service process lpc servicing thread mark ready priority temporarily boost avoid scheduling delay occur wait thread queue windows xp improve performance reduce code path length critical function well algorithm processor datum structure memory coloring non uniform memory access numa machine implement scalable locking protocol queue spinlock new locking protocol help reduce system bus cycle include lock free list queue atomic read modify write operation like interlock increment advanced synchronization technique time windows 7 develop major change come computing client server computing increase importance advanced local procedure alpc facility introduce provide high performance reliability lpc number cpu physical memory available large multiprocessor increase substantially lot effort improve implementation smp windows nt bitmask represent collection processor identify example set processor particular thread schedule bitmask define fit single word memory limit number processor support system 64 windows 7 add concept processor group represent arbitrary number cpu accommodate cpu core number cpu core single system continue increase core core support logical thread execution time additional cpu create great deal contention lock schedule cpu memory windows 7 break lock apart example windows 7 single lock windows scheduler synchronize access queue contain thread wait event windows 7 object lock allow queue access concurrently execution path scheduler rewrite lock free change result good scalability performance windows system 256 hardware

free change result good scalability performance windows system 256 hardware thread change increase importance support par- allel

computing year computer industry dominate moore law lead high density transistor manifest them- self fast clock rate cpu moore law continue hold true limit reach prevent cpu clock rate increase instead transistor build cpu chip new programming model achieve parallel execution microsoft concurrency runtime concrt intel threading building blocks tbb express parallelism c++ program moore law govern computing year amdahl law govern parallel computing rule future support task base parallelism windows 7 provide new form user mode scheduling ums ums allow program decompose task task schedule available cpu scheduler operate user mode kernel advent multiple cpu small computer shift take place parallel computing graphic process unit gpu accelerate computational algorithm need graphic simd architecture execute single instruction multiple datum time give rise use gpu general computing graphic operate system support software like opencl cuda allow program advantage gpu windows support use gpu software directx graphic support software call directcompute allow program specify computational kernel hlsl high level shader language program model program simd hardware graphic shader computational kernel run quickly gpu return result main computation run cpu extensibility refer capacity operate system advance compute technology facilitate change time devel- oper implement windows layered architecture windows exec- utive run kernel mode provide basic system service abstrac- tion support share use system executive server subsystem operate user mode environmental sub- system emulate different operating system program write win32 api posix run windows appropriate environment modular structure additional environmental subsystem add affect executive addition windows use loadable driver o system new file system new kind o device new kind networking add system run windows use client server model like mach operate system support dis- tribute processing remote procedure call rpcs define open operate system portable move cpu architec- ture change windows design portable like unix operate system windows write primarily c c++ architecture

like unix operate system windows write primarily c c++ architecture specific source code relatively small little use assembly code port windows new architecture affect windows kernel user mode code windows exclu- sively write architecture independent port windows kernel architecture specific code port conditional compi- lation need part kernel change major data structure page table format entire windows system recompile new cpu instruction set operate system sensitive cpu architecture cpu support chip hardware boot program cpu support chip collectively know chipset chipset associate boot code determine interrupt deliver describe physical characteristic system provide interface deep aspect cpu architecture error recovery power management burdensome port windows type support chip cpu architecture instead windows isolate chipset dependent code dynamic link library dll call hardware abstraction layer hal load kernel windows kernel depend hal interface underlie chipset detail allow single set kernel driver binary particular cpu different chipset simply load different version hal year windows port number different cpu architecture intel ia-32 compatible 32 bit cpu amd64 compatible ia64 64 bit cpu dec alpha mips powerpc cpu cpu architecture fail market windows 7 ship ia- 32 amd64 architecture support client computer amd64 ia64 server windows design international multinational use provide support different locale national language support nls api nls api provide specialize routine format date time money accordance national custom string comparison specialize account vary character set unicode windows native character code windows support ansi character convert unicode character manipulate 8 bit 16 bit conversion system text string keep resource file replace localize system different language multiple locale concurrently important multilingual individual business increase energy efficiency computer cause battery long laptop netbook save significant operating cost power cooling datum center contribute green initiative aim lower energy consumption business consumer time windows implement strategy decrease energy use cpu move low power state example lower clock

energy use cpu move low power state example lower clock frequency possible addition computer actively windows entire computer low power state sleep save memory disk shut computer hibernation user return computer power continue previous state user need reboot restart application windows 7 add new strategy save energy long cpu stay unused energy save computer fast human being lot energy save human think problem program constantly poll happen system swarm software timer fire keep cpu stay idle long save energy windows 7 extend cpu idle time skip clock tick coalesce software timer small number event park entire cpu system dynamic device support early history pc industry computer configuration fairly static occasionally new device plug serial printer game port computer step dynamic configuration pc laptop dock pcmia card pc suddenly connect disconnect set peripheral contemporary pc situation completely change pc design let user plug unplug huge host peripheral time external disk thumb drive camera like constantly come go support dynamic configuration device continually evolve windows system automatically recognize device hardware abstraction layer windows block diagram plug find install load appropriate driver with- user intervention device unplug driver automatically unload system execution continue disrupt software architecture windows layered system module show fig- ure b.1 main layer hal kernel executive run kernel mode collection subsystem service run user mode user mode subsystem fall category environmental subsystem emulate different operating system protection subsystem provide security function chief advantage type architecture interaction module keep simple remainder section describe layer subsystem hal layer software hide hardware chipset difference upper level operate system hal export virtual hard- ware interface kernel dispatcher executive device driver single version device driver require cpu architecture matter support chip present device driver map device access directly chipset specific detail map memory configure o bus set dma cope motherboard specific facility provide hal interface bus set dma cope motherboard specific facility provide hal interface kernel layer windows main responsibility thread scheduling low level processor synchronization interrupt exception handling switch user mode kernel mode kernel implement c language assembly language absolutely necessary interface low level hardware architecture kernel organize accord object orient design principle object type windows system define data type set attribute data value set method example function operation object instance object type kernel perform job set kernel object attribute store kernel datum method perform kernel activity kernel dispatcher provide foundation executive sub- system dispatcher page memory execu- tion preempt main responsibility thread scheduling context switching implementation synchronization primitive timer man- agement software interrupt asynchronous deferred procedure call exception dispatching thread scheduling like modern operate system windows use process thread executable code process thread thread scheduling state include actual priority processor affinity cpu usage information possible thread state ready standby run waiting transition terminate ready indicate thread wait run high priority ready thread move standby state mean thread run multiprocessor system processor keep thread standby state thread run execute processor run preempt high priority thread terminate allot execution time quantum end wait dispatcher object event signal o completion thread waiting state wait dispatcher object signal thread transition state wait resource necessary execution example wait kernel stack swap disk thread enter terminate state finish execution dispatcher use 32 level priority scheme determine order thread execution priority divide class variable class real- time class variable class contain thread have priority 1 15

real time class contain thread priority range 16 31 dispatcher use queue scheduling priority traverse set queue high low find thread ready run thread particular processor affinity processor available dispatcher skip past continue look ready thread willing run available processor ready thread find dispatcher execute special thread call idle thread priority class 0 reserve idle thread thread time quantum run clock interrupt queue quantum end defer procedure dpc processor queue dpc result software interrupt processor return normal interrupt priority software interrupt cause dispatcher reschedule processor execute available thread preempt thread priority preempt thread modify place dispatcher queue preempt thread variable- priority class priority lower priority lower base priority lower thread priority tend limit cpu consump- tion

compute bind thread versus o bind thread variable- priority thread release wait operation dispatcher boost priority boost depend device thread wait example thread wait keyboard o large priority increase thread wait disk operation moderate strategy tend good response time interactive thread mouse window enable o bind thread o device busy permit compute bind thread use spare cpu cycle background addition thread associate user active gui window receive priority boost enhance response time scheduling occur thread enter ready wait state thread terminate application change thread priority pro- cessor affinity high priority thread ready low priority thread run low priority thread preempt preemption give high priority thread preferential access cpu windows hard real time operate system guarantee real time thread start execute particular time limit thread block indefinitely dpcs interrupt service routine isrs run discuss traditionally operate system scheduler sampling measure cpu utilization thread system timer fire periodically timer interrupt handler note thread currently schedule execute user kernel mode interrupt occur sampling technique necessary cpu high resolution clock clock expensive unreliable access frequently efficient sampling inaccurate lead anomaly incorporate interrupt servicing time thread time dispatch thread run fraction quantum start windows vista cpu time windows track hardware timestamp counter tsc include recent processor tsc result accurate accounting cpu usage scheduler preempt thread run quantum implementation synchronization primitive key operating system data structure manage object common facility allocation reference counting security dispatcher object control dispatching synchronization system example object include follow event object record event occurrence synchronize occurrence action notification event signal wait thread synchronization event signal single wait thread mutant provide kernel mode user mode mutual exclusion associ- ate notion ownership mutex available kernel mode provide deadlock free mutual semaphore object act counter gate control number thread access resource thread object entity schedule kernel dispatcher associate process object encapsulate virtual address space

schedule kernel dispatcher associate process object encapsulate virtual address space thread object signal thread exit process object process exit timer object track time signal timeout operation long need interrupt periodic activity need schedule dispatcher object access user mode open operation return handle user mode code poll wait handle synchronize thread operate system software interrupt asynchronous deferred procedure call dispatcher implement type software interrupt asynchronous procedure call apcs deferred procedure call dpcs mention early asynchronous procedure break execute thread call procedure apcs begin execution new thread suspend resume exist thread terminate thread process deliver notification asynchronous o complete extract content cpu register run thread apcs queue specific thread allow system execute system user code process context user mode execution apc occur arbitrary time thread wait kernel mark alertable dpcsare postpone interrupt processing handle urgent device interrupt processing isr schedule remain processing queue dpc associate software interrupt occur cpu priority low priority o device interrupt high priority thread run dpc block device isr addition defer device interrupt processing dispatcher use dpc process timer expiration preempt thread execution end scheduling quantum execution dpc prevent thread schedule current processor keep apc signal completion o. completion dpc routine extended time alternative dispatcher maintain pool worker thread isr dpc queue work item worker thread execute normal thread scheduling dpc routine restrict page fault page memory system service action result attempt wait dispatcher object signal unlike apcs dpc routine assumption process context processor execute exception interrupt kernel dispatcher provide trap handling exception interrupt generate hardware software windows define architecture independent exception include memory access violation integer overflow float point overflow underflow integer divide zero float point divide zero illegal instruction data misalignment privileged instruction page read error access violation paging file quota exceed debugger breakpoint debugger single step trap handler deal simple exception elaborate exception handling perform

step trap handler deal simple exception elaborate exception handling perform kernel exception dispatcher exception dispatcher create exception record contain reason exception find exception handler deal exception occur kernel mode exception dispatcher simply call routine locate exception handler handler find fatal system error occur user leave infamous blue screen death signify system failure exception handling complex user mode process environmental subsystem posix system set debugger port exception port process create detail port section b.3.3.4 debugger port register exception handler send exception port debugger port find handle exception dispatcher attempt find appropriate exception handler handler find debugger call catch error debugging debugger run message send process exception port environmental subsystem chance translate exception example posix environment translate windows exception message posix signal send thread cause exception finally work kernel simply terminate process contain thread cause exception windows fail handle exception construct description error occur request permission user send information microsoft analysis case microsoft automated analysis able recognize error immediately sug- gest fix workaround interrupt dispatcher kernel handle interrupt call interrupt service routine isr supply device driver kernel trap- handler routine interrupt represent interrupt object con- tain information need handle interrupt interrupt object make easy associate interrupt service routine interrupt have access interrupt hardware directly different processor architecture different type number inter- rupt portability interrupt dispatcher map hardware interrupt standard set interrupt prioritize service prior- ity order 32 interrupt request level irql windows reserve use kernel remain 24 represent hardware interrupt hal(although ia-32 system use 16 windows interrupt define figure b.2 kernel use interrupt dispatch table bind interrupt level service routine multiprocessor computer windows keep separate interrupt dispatch table idt processor processor irql set independently mask interrupt interrupt occur level equal irqlof processor block irqli lower type interrupt machine check bus error clock track time traditional pc irq hardware interrupt dispatch deferred procedure

track time traditional pc irq hardware interrupt dispatch deferred procedure dpc kernel asynchronous procedure apc interprocessor notification request processor act e.g. dispatch process update tlb windows interrupt request level kernel level thread isr return interrupt processing windows take advantage property use software interrupt deliver apcs dpc perform system function synchronize thread o completion start thread execution handle timer switch user mode kernel mode thread programmer think thread traditional windows actually thread user mode thread ut

kernel mode thread kt stack register value execution context ut request system service execute instruction cause trap kernel mode kernel layer run trap handler switch ut correspond kt kt complete kernel execution ready switch correspond ut kernel layer call switch ut continue execution user mode windows 7 modify behavior kernel layer support user- mode scheduling uts user mode scheduler windows 7 support cooperative scheduling ut explicitly yield ut call user mode scheduler necessary enter kernel user mode scheduling

explain detail section b.7.3.7 windows executive provide set service environmental sub- system use service group follow object manager virtual memory manager process manager advanced local procedure facility o manager cache manager security reference monitor plug play power manager registry booting manage kernel mode entity windows use generic set interface manipulate user mode program windows call entity object executive component manipulate object manager example object semaphore mutexe event process thread dispatcher object thread block ker- nel dispatcher wait object signal process thread virtual memory api use process thread handle identify process thread operate example object include file section port internal o object file object maintain open state file device section map file local communication endpoint implement port object user mode code access object opaque value call handle return api process handle table contain entry track object process system pro- cess contain kernel handle table protect user code handle table windows represent tree struc- ture expand hold 1,024 handle hold 16 million kernel mode code access object handle reference aprocess get handle create object open exist object receive duplicated handle process inherit handle parent process process exit open handle implicitly close object manager entity generate object handle natural place check security object manager check process right access object process try open object object manager enforce quota maximum memory process use charge process memory occupy reference object refuse allocate memory accumulate charge exceed process quota object manager keep track count object number handle object number reference pointer handle count number handle refer object handle table process include system process contain kernel reference pointer count incremente new pointer need kernel decremente kernel pointer purpose reference count ensure object free reference handle internal kernel pointer object manager maintain windows internal space con- trast unix root system space file system windows use abstract space connect file

system space file system windows use abstract space connect file system device windows object creator process thread create name reference handle separate numerical identifier synchronization event usually name open unrelated process permanent temporary permanent represent entity disk drive remain process access temporary exist process hold handle object object manager support directory symbolic link space example ms dos drive letter implement symbolic link global??c symbolic link device object devicehardiskvolume2 represent mount file system volume device directory object mention early instance object type object type specify instance allocate data field define standard set virtual function object implement standard function implement operation map name object close delete apply security check function specific particular type object implement system service design operate particular object type method specify object type parse function interesting standard object func- tion allow implementation object file system registry configuration store gui object notable user parse func- tion extend windows space return windows naming example device object rep- resent file system volume provide parse function allow like global??c:foobar.doc

interpret file foobar.doc volume represent device object harddiskvolume2 illustrate naming parse function object handle work look step open file windows 1 application request file name c:foobar.doc open 2 object manager find device object harddiskvolume2 look parse procedure iopparsedevice object type invoke file relative root file system 3 iopparsedevice allocate file object pass file system fill detail access c:foobar.doc volume 4 file system return iopparsedevice allocate entry file object handle table current process return handle application file successfully open iopparsedevice

delete file object allocate return error indication application virtual memory manager executive component manage virtual address space physi- cal memory allocation paging virtual memory vm manager design vm manager assume underlie hardware sup- port virtual physical mapping paging mechanism transparent cache coherence multiprocessor system allow multiple page table entry map physical page frame vm manager windows use page base management scheme page size 4 kb 2 mb amd64 ia-32 compatible processor 8 kb ia64 page datum allocate process physical memory store paging file disk map directly regular file local remote file system page mark zero fill demand initialize page zero allocate erase previous content ia-32 processor process 4 gb virtual address space upper 2 gb identical process windows kernel mode access operate system code datum structure amd64 architecture windows provide 8 tb virtual address space user mode 16 eb support exist hardware process key area kernel mode region identical process self map hyperspace session space hardware reference process page table physical page frame number page table self map make content process page table accessible virtual address hyperspace map current process work set information kernel mode address space session space share instance win32 session specific driver process terminal server ts session different ts session share different instance driver map virtual address low user mode region virtual address space specific process accessible user- kernel mode thread windows vm manager use step process allocate virtual memory step reserve page virtual address process virtual address space second step commit allocation assign virtual memory space physical memory space paging file windows limit virtual memory space process consume enforce quota commit memory process decommit memory long free virtual memory space use process api reserve virtual address commit virtual memory handle process object parameter allow process control virtual memory environmental subsystem manage memory client process way windows

virtual memory environmental subsystem manage memory client process way windows implement share memory define section object get handle section object process map memory section range address call view process establish view entire section portion need windows allow section map current process process caller section way section back disk space system page file regular file memory map file section base mean appear virtual address process attempt access section represent physical memory allow 32 bit process access physical memory fit virtual address space finally memory protection page section set read read write read write execute execute access let look closely protection setting access page raise exception access exception example check faulty program iterate end array simply detect program attempt access virtual address commit memory user- kernel mode stack use access page guard page detect stack overflow use look heap buffer overrun user- mode memory allocator special kernel allocator device verifier configure map allocation end page follow access page detect programming error access end allocation copy write mechanism enable vm manager use physical memory efficiently process want independent copy datum section object vm manager place single shared copy virtual memory activate copy write property region memory process try modify datum copy- write page vm manager make private copy page virtual address translation windows use multilevel page table ia-32 amd64 processor process page directory con- tain 512 page directory entry pdes 8 byte size pde point pte table contain 512 page table entry ptes 8 byte size pte point 4 kb page frame physical memory variety reason hardware require page directory pte table level multilevel page table occupy single page number pde pte fit page determine virtual address translate page figure b.3 diagram structure structure describe far represent 1 gb virtual address translation ia-32 second page directory level need con- page directory pointer table taine entry show diagram 64 bit processor level need amd64 windows use total level total size page table page need fully represent 32 bit virtual address space process 8 mb vm manager allocate page pde pte need move page table page disk use page table page fault memory reference consider virtual address translate physical address ia-32 compatible processor 2 bit value represent value 0 1 2 3 9 bit value represent value 0 511 12 bit value value 0 4,095 12 bit value select byte 4 kb page memory 9 bit value represent 512 pde pte page directory pte table page show figure b.4 translate virtual address pointer byte address physical memory involve break 32 bit pointer value start significant bit bit index pde level page table select pde contain physical page number page directory page map 1 gb address space virtual physical address translation ia-32 bit

select pde time second level page directory pde contain physical page number 512 bit select 512 pte select pte table page select pte contain physical page number byte access bit byte offset page physical address byte access construct append low 12 bit virtual address end physical page number find select pte number bit physical address different number bit virtual address original ia-32 architecture pte pde 32 bit structure room 20 bit physical page number physical address size virtual address size system address 4 gb physical memory later ia-32 extend large 64 bit pte size today hardware support 24 bit physical address system support 64 gb server system today windows server base amd64 ia64 support large physical address possibly use course time 4 gb optimistically large improve performance vm manager map page directory pte table page contiguous region virtual address process self map allow vm manager use pointer access current pde pte correspond particular virtual address matter process run self map ia-32 take contiguous 8 mb region kernel virtual address space amd64 self map occupy 512 gb self map occupy significant address space require additional virtual memory page allow page table page automatically page physical memory creation self map pde level page directory refer page directory page form loop page table translation virtual page access loop take pte table page access loop take low level page directory page access loop take twice forth additional level page directory 64 bit virtual memory translate way virtual address pointer break value amd64 windows use level map 512 page 9 + 9 + 9 + 9 + 12 = 48 bit virtual address avoid overhead translate virtual address look pde pte processor use translation look aside buffer tlb hardware contain associative memory cache map virtual page pte tlb memory management unit mmu processor mmu need

page pte tlb memory management unit mmu processor mmu need walk navigate data structure page table memory need translation miss tlb pdes ptes contain physical page number bit reserve operating system use bit control hardware use memory hardware caching page addition entry specify kind access allow user kernel mode pde mark function pte pde ia-32 11 bit virtual address pointer select pde level translation select pde mark act pte remain 21 bit pointer offset byte result 2 mb size page mix match 4 kb 2 mb page size page table easy operate system significantly improve performance program reduce mmu need reload entry tlb pde map 2 mb replace 512 pte map 4 kb manage physical memory 2 mb page available need difficult continually break 4 kb page cause external fragmentation memory large page result significant internal fragmentation problem typically windows large server application use large page improve performance tlb well suited operating system server application start run system boot memory fragment windows manage physical memory associate physical page seven state free zeroed modified standby bad transition free page page particular content zeroed page free page zero ready immediate use satisfy zero demand fault modified page write process send disk allocate process standby page copy information store disk standby page page modify modify page write disk page prefetche expect soon bad page unusable hardware error detect transition page way disk page frame allocate valid page work set process contain process page table valid page contain process page table page state keep separate list accord state type list construct link correspond entry page frame number pfn database include entry physical memory page pfn entry include information reference count lock numa information note pfn database represent page physical memory pte represent page virtual memory valid bit pte zero hardware ignore bit vm manager

memory valid bit pte zero hardware ignore bit vm manager define use invalid page number state represent bit pte page file page page file offset page file page table entry valid bit zero fault mark zero demand page map section object encode pointer appropriate section object pte page write page file contain information locate page disk forth structure page file pte show figure b.5 t p v bit zero type pte pte include 5 bit page protection 32 bit page file offset 4 bit select paging file 20 bit reserve additional bookkeeping windows use working set recently lru replacement policy page process appropriate process start assign default minimum work set size work set process allow grow remain physical memory start run low point vm manager start track age page work set eventually available memory run critically low vm manager trim work set remove old page old page depend long memory reference determine periodically make pass working set process incremente age page mark pte reference pass necessary trim work set vm manager use heuristic decide trim process remove old page process working set trim plenty memory available give hard limit physical memory use windows 7 vm manager trim process grow rapidly memory plentiful policy change significantly improve responsiveness system process windows track working set user mode process system process include pageable data structure code run kernel mode windows 7 create additional work set system process associate particular category kernel memory file cache kernel heap kernel code work set distinct work set allow vm manager use different policy trim different category kernel memory vm manager fault page immediately need research show memory reference thread tend locality property page likely adjacent page reference near future think iterate array fetch sequential instruction form executable code thread locality vm manager fault page fault adjacent page

code thread locality vm manager fault page fault adjacent page prefetching tend reduce total number page fault allow read cluster improve o performance addition manage commit memory vm manager manage process reserve memory virtual address space process associate tree describe range virtual address use use allow vm manager fault page table page need pte faulting address uninitialized vm manager search address process tree virtual address descriptor vads use information fill pte retrieve page case pte- table page exist page transparently allocate initialize vm manager case page share section object vad contain pointer section object section object contain information find share virtual page pte initialize point directly windows process manager provide service create delete process thread job knowledge parent child relationship process hierarchy refinement leave particular environmental subsystem own process process manager involve scheduling process set priority affinity process thread create thread scheduling take place kernel dispatcher process contain thread process collect large unit call job object use job object allow limit place cpu usage work set size processor affinity control multiple process job object manage large example process creation win32 environment follow 1 win32 application call createprocess 2 message send win32 subsystem notify process 3 createprocess original process call api process manager nt executive actually create process 4 process manager call object manager create process object return object handle win32 5 win32 call process manager create thread process return handle new process thread windows api manipulate virtual memory thread duplicate handle process handle subsystem perform operation behalf new process have execute directly new process context new process create initial thread create asynchronous procedure deliver thread prompt start execution user mode image loader loader ntdll.dll link library automatically map newly create process windows support unix fork style process creation order support posix environmental subsystem win32 environment call process manager directly client

posix environmental subsystem win32 environment call process manager directly client process posix use cross process nature windows api create new process subsystem process process manager rely asynchronous procedure call apcs implement kernel layer apc initiate thread execution suspend resume thread access thread register terminate thread process support debugger debugger support process manager include api suspend resume thread create thread begin suspend mode process manager api set thread register context access process virtual memory thread create current process inject process debugger make use thread injection execute code process debug run executive thread temporarily attach different process thread attach kernel worker thread need execute context process originate work request example vm manager use thread attach need access process working set page table o manager use update status variable process asynchronous o operation process manager support impersonation thread associate security token login process authenticate user security token attach user process inherit child pro- cesses token contain security identity sid user sid group user belong privilege user integrity level process default thread process share common token represent user application start process thread run process security token belong user set thread specific token belong user impersonate user impersonation facility fundamental client server rpc model service act behalf variety client different security id right impersonate user deliver rpc connection client process server process impersonation allow server access system service client order access create object file behalf client server process trustworthy carefully write robust attack client server process impersonate user subsequent client request facility client server computing implementation windows use client server model environmental subsystem server implement particular operating- system personality service user authentication net- work facility printer spooling web service network file system plug- play implement model reduce memory foot- print multiple service collect process run svchost.exe program service load dynamic link library dll implement

run svchost.exe program service load dynamic link library dll implement service rely user mode thread pool facili- tie share thread wait message section b.3.3.3 normal implementation paradigm client server computing use rpc communicate request win32 api support standard rpc pro- tocol describe section b.6.2.7 rpc use multiple transport example name pipe tcp ip implement rpc system rpc occur client server local system advanced local procedure facility alpc transport low level system implementation environmental system service available early stage booting rpc available instead native windows service use alpc directly alpc message passing mechanism server process publish globally visible connection port object client want service subsystem service open handle server connection port object send connection request port server create channel return handle client channel consist pair private com- munication port client server message server- client message communication channel support callback mechanism client server accept request normally expect reply alpc channel create message passing technique 1 technique suitable small medium message 63 kb case port message queue intermediate storage message copy process 2 second technique large message case shared- memory section object create channel message send port message queue contain pointer size information refer section object avoid need copy large message sender place datum share section receiver view 3 technique use api read write directly process address space alpc provide function synchronization server access datum client technique normally rpc achieve high performance specific scenario win32 window manager use form message passing independent executive alpc facility client ask connec- tion use window manager messaging server set object 1 dedicated server thread handle request 2 64 kb shared section object 3 event pair object event pair object synchronization object win32 subsystem provide notification client thread copy message win32 server vice versa section object pass message event pair object provide synchronization window manager messaging advantage section

event pair object provide synchronization window manager messaging advantage section object eliminate message copying represent region share memory event pair object eliminate overhead port object pass message contain pointer length dedicated server thread eliminate overhead determine client thread call server server thread client kernel give scheduling preference dedicated server thread improve performance o manager responsible manage file system device driver network driver keep track device driver filter driver file system load manage buffer o request work vm manager provide memory map file o control windows cache manager handle caching entire o system o manager fundamentally asynchronous provide synchronous o explicitly wait o operation complete o manager provide model asynchronous o completion include setting event updating status variable call process delivery apc initiate thread use o completion port allow single thread process o completion thread device driver arrange list device call driver o stack driver represent system driver object single driver operate multiple device driver represent o stack device object contain link driver object o manager convert request receive standard form call o request packet irp forward irp driver target o stack processing driver process irp call o manager forward irp driver stack processing finish complete operation represent irp o request complete context different example driver perform o operation force block extended time queue irp worker thread continue process system context original thread driver return status indicate o request pende thread continue execute parallel o operation irp process interrupt service routine complete arbitrary process context final processing need place context initiate o o manager use apc final o completion processing process context originate thread o stack model flexible driver stack build vari- ous driver opportunity insert stack filte driver filter driver examine potentially modify o operation mount management partition management disk striping mirroring example functionality implement filter driver execute beneath file

striping mirroring example functionality implement filter driver execute beneath file system stack file system filter driver execute file system implement functionality hier- archical storage management single instancing file remote boot dynamic format conversion party use file system filter driver implement virus detection device driver windows write windows driver model wdm specification model lay requirement device driver include layer filter driver share common code han- dling power plug play request build correct cancellation logic richness wdm write wdm device driver new hardware device involve great deal work fortunately port miniport model make unnecessary class similar device audio driver sata device ethernet controller instance device share common driver class call port driver port driver implement standard operation class call device specific routine device miniport driver imple- ment device specific functionality tcp ip network stack implement way ndis.sys class driver implement network driver functionality call network miniport driver specific recent version windows include windows 7 provide additional simplification write device driver hardware device kernel mode driver write kernel mode driver framework kmdf provide simplified programming model driver wdm option user mode driver framework umdf driver need operate kernel mode easy develop deploy driver user mode make system reliable failure user mode driver cause kernel mode crash operate system caching file system instead win- dow provide centralized caching facility cache manager work closely vm manager provide cache service component control o manager caching windows base file raw block size cache change dynamically accord free memory available system cache manager maintain pri- vate working set share system process working set cache manager memory map file kernel memory use special interface vm manager fault page trim private cache divide block 256 kb cache block hold view memory map region file cache block describe virtual address control block vacb store virtual address file offset view number process block vacb store virtual address file offset view number process view vacbs reside single array maintain cache manager o manager receive file user level read request o manager send irp o stack volume file reside file mark cacheable file system call cache manager look request datum cache file view cache manager calculate entry file vacb index array correspond byte offset request entry point view cache invalid invalid cache manager allocate cache block corresponding entry vacb array map view cache block cache manager attempt copy datum map file caller buffer copy succeed operation complete copy fail page fault cause vm manager send noncached read request o manager o manager send request driver stack time request paging operation bypass cache manager read datum file directly page allocate cache manager completion vacb set point page datum cache copy caller buffer original o request complete figure b.6 show overview operation akernel level read operation similar datum access directly cache copy buffer user space use file system metadata data structure describe file system kernel use cache manager mapping interface read metadata modify metadata file system use cache manager pin interface pin page lock page physical memory page frame vm manager page page update metadata file system ask cache manager unpin page amodified page mark dirty vm manager flush page disk improve performance cache manager keep small history read request history attempt predict future request cache manager find pattern previous request sequential access forward backward prefetche datum cache request submit application way application find datum cache need wait disk o. cache manager responsible tell vm manager flush content cache cache manager default behavior write caching accumulate write 4 5 second wake cache- writer thread write caching need process set flag open file process explicit cache flush function fast write process potentially fill free cache page cache

function fast write process potentially fill free cache page cache writer thread chance wake flush page disk cache writer prevent process flood system following way free cache memory low cache manager temporarily block process attempt write datum wake cache- writer thread flush page disk fast write process actually network redirector network file system block long cause network transfer time retransmit retransmission waste network bandwidth prevent waste network redirector instruct cache manager limit backlog write cache network file system need datum disk network interface cache manager provide dma interface datum directly move datum directly avoid need copy datum intermediate buffer security reference monitor centralize management system entity object manager enable windows use uniform mechanism perform run time access validation audit check user accessible entity system process open handle object security reference monitor srm check process security token object access control list process necessary access right srm responsible manipulate privilege security token special privilege require user perform backup restore operation file system debug process forth token mark restrict privilege access object available user restricted token primarily limit damage execution untrusted code integrity level code execute process represent token integrity level type capability mechanism mention early aprocess modify object integrity level high code execute process permission grant integrity level introduce hard code successfully attack outward face software like internet explorer system responsibility srm log security audit event department defense common criteria 2005 successor orange book require secure system ability detect log attempt access system resource easily trace attempt unauthorized access srm responsible make access check generate audit record security event log operate system use plug play pnp manager recognize adapt change hardware configuration pnp device use standard protocol identify system pnp manager automatically recognize instal device detect change device system operate manager keep track hardware resource device potential resource take care load appropriate

track hardware resource device potential resource take care load appropriate driver management hardware resource primarily interrupt o memory range goal determine hardware configuration device able operate successfully pnp manager handle dynamic reconfiguration follow get list device bus driver example pci usb load instal driver find necessary send add device request appropriate driver device pnp manager figure optimal resource assignment send start device request driver specify resource assignment device device need reconfigure pnp manager send query stop request ask driver device temporarily disable driver disable device pending operation complete new operation prevent start finally pnp manager send stop request reconfigure device new start device request pnp manager support request example query- remove operate similarly query stop employ user get ready eject removable device usb storage device surprise remove request device fail likely user remove device tell system stop finally remove request tell driver stop device permanently program system interested addition removal device

pnp manager support notification notification example give gui file menu information need update list disk volume new storage device attach remove instal device result add new service svchost.exe process system service frequently set run system boot continue run original device plug system windows 7 introduce service trigger mechanism service control manager scm manage system service mechanism service register start scm receive notification pnp manager device interest add system windows work hardware implement sophisticated strategy energy efficiency describe section b.2.8 policy drive strategy implement power manager power manager detect current system condition load cpu o device improve energy efficiency reduce performance responsiveness system need low power manager entire system efficient sleep mode write content memory disk turn power allow system hibernation primary advantage sleep system enter fairly quickly second lid close laptop return sleep fairly quick power turn low cpu o device memory continue power content hibernation take considerably long entire content mem- ory transfer disk system turn fact system fact turn significant advantage loss power system battery swap lap- desktop system unplug save system datum lose unlike shutdown hibernation save currently run system user resume leave hibernation require power system remain hibernation indefinitely like pnp manager power manager provide notification rest system change power state application want know system shut start save state disk windows keep configuration information internal database call hive manage windows configuration manager commonly know registry separate hive system information default user preference software installation security boot option information system hive require boot system registry manager implement component executive registry represent configuration state hive hierarchical namespace key directory contain set type value unicode string ansi string integer untyped binary datum theory new key value create initialize new software instal modify reflect change configuration software practice registry general purpose database interprocess communication mechanism inventive restart application system time

purpose database interprocess communication mechanism inventive restart application system time configuration change nuisance instead program rely kind notification provide pnp power manager learn change system configuration registry supply notification allow thread register notify change registry thread detect adapt configuration change record registry significant change system update operate system driver instal danger configuration datum corrupt example work driver replace nonworking driver application fail install correctly leave partial information

registry windows create system restore point make change restore point contain copy hive change return version hive corrupted system work improve stability registry configuration windows add transaction mechanism begin windows vista prevent registry partially update collection related configuration change registry transaction general trans- action administer kernel transaction manager ktm include file system transaction ktm transaction seman- tic find normal database transaction supplant system restore facility recover damage registry configuration cause software installation booting windows pc begin hardware power firmware begin execute rom old machine firmware know bios modern system use uefi unified extensible firmware interface fast general make well use facility contemporary processor firmware run power self- test post diagnostic identify device attach system initialize clean power state build description advanced configuratio power interface acpi firmware find system disk load windows bootmgr program begin execute machine hibernate winresume program load restore run system disk system continue execu- tion point reach right hibernate machine shut bootmgr perform initialization system load winload program load hal.dll kernel ntoskrnl.exe driver need booting system hive winload transfer execution kernel kernel initialize create process system pro- cess contain internal kernel worker thread execute user mode user mode process create smss session manager subsystem similar init initialization process unix smss perform initialization system include establish paging file load device driver manage windows session session represent log user session 0 run system wide background service lsass services session anchor instance csrss process session 0 initially run winlogon process process log user launch explorer process implement windows gui experience following list itemize aspect booting smss complete system initialization start session 0 login session wininit run session 0 initialize user mode start lsass services local session manager lsm lsass security subsystem implement facility authentication services contain service control manager scm supervise background activity system include user mode service number service

supervise background activity system include user mode service number service register start system boot start demand trigger event arrival device csrss win32 environmental subsystem process start session unlike posix subsystem start demand posix process create winlogon run windows session session 0 log system optimize boot process prepage file disk base previous boot system disk access pattern boot lay system file disk reduce number o operation require process necessary start system reduce group service few process approach contribute dramatic reduction system boot time course system boot time important sleep hibernation capability windows terminal services fast user switching windows support gui base console interface user key- board mouse display system support audio video audio input windows voice recognition software voice recognition make system convenient increase accessibility user dis- ability windows 7 add support multi touch hardware allow user input datum touch screen make gesture finger eventually video input capability currently com- munication application likely visually interpret gesture microsoft demonstrate xbox 360 kinect product future input experience evolve microsoft surface computer instal public venue hotel conference center surface computer table surface special camera underneath track action multiple user recognize object place pc course envision personal computer inherently single user machine modern windows support sharing pc multiple user user log gui session create represent gui environment contain process create run application windows allow multiple session exist time single machine windows support single console consist monitor keyboard mouse connect pc session connect console time logon screen display console user create new session attach exist session previously create allow multiple user share single pc have log user microsoft call use session fast user switching user create new session connect exist session pc session run windows pc terminal server ts connect gui window user local session new existing session call remote desktop remote computer common use remote

existing session call remote desktop remote computer common use remote desktop user connect session work pc home pc corporation use corporate terminal server system maintain datum center run user session access corporate resource allow user access resource pc user office server computer handle dozen remote desktop session form thin client computing individual computer rely server function rely data center terminal server improve reliability manageability security corporate compute resource ts windows implement remote assistance remote user invite share session user log session console remote user watch user action give control desktop help resolve computing problem native file system windows ntfs local volume associated usb thumb drive flash memory camera external disk format 32 bit fat file system portability fat old file system format understand system windows software run camera disadvantage fat file system restrict file access authorized user solution secure datum fat run application encrypt datum store file system contrast ntfs use acls control access individual file sup- port implicit encryption individual file entire volume windows bitlocker feature ntfs implement feature include data recovery fault tolerance large file file system multiple data stream unicode name sparse file journaling volume shadow copy ntfs internal layout fundamental entity ntfs volume volume create win- dows logical disk management utility base logical disk partition avolume occupy portion disk entire disk span ntfs deal individual sector disk instead use clus- ter unit disk allocation cluster number disk sector power 2 cluster size configure ntfs file system format- te

default cluster size base volume size—4 kb volume large 2 gb give size today disk sense use cluster size large windows default achieve well performance performance gain come expense

internal ntfs use logical cluster number lcns disk address assign number cluster beginning disk end scheme system calculate physical disk offset byte multiply lcn cluster size file ntfs simple byte stream unix structured object consist type attribute attribute file independent byte stream create delete read write attribute type standard file include file name file alias ms dos short creation time security descriptor specify access control list user datum store traditional data file unnamed datum attribute contain file datum additional data stream create explicit name instance macintosh file store windows server resource fork name data stream iprop interface component object model com use name datum stream store property ordinary file include thumbnail image general attribute add necessary access file attribute syntax ntfs return size unnamed attribute response file query operation run dir command file ntfs describe record array store special file call master file table mft size record determine file system create range 1 4 kb small attribute store mft record call resident attribute large attribute unnamed bulk datum call nonresident attribute store contiguous extent disk pointer extent store mft record small file datum attribute fit inside mft record file attribute highly fragment pointer need point fragment record mft large case file describe record call base fil record contain pointer overflow record hold additional pointer attribute file ntfs volume unique id call fil reference file reference 64 bit quantity consist 48 bit file number 16 bit sequence number file number record number array slot mft describe file sequence number incremente time mft entry reuse sequence number enable ntfs perform internal consistency check catch stale reference delete file mft entry reuse new file ntfs b+ tree unix ntfs namespace organize hierarchy directory directory use data structure call b+ tree store index file name directory b+ tree length path root tree leaf cost

name directory b+ tree length path root tree leaf cost reorganize tree eliminate index root directory contain level b+ tree large directory level contain pointer disk extent hold remainder tree entry directory contain file reference file copy update timestamp file size take file resident attribute mft copy information store directory directory listing efficiently generate file name size update time available directory need gather attribute mft entry file ntfs volume metadata store file file mft second file recovery mft damage contain copy 16 entry mft file special purpose include file describe log file record metadata update file system volume file contain volume version ntfs format volume bit tell volume corrupt need check consistency chkdsk attribute definitio table indicate attribute type volume operation perform root directory level directory file system hierarchy bitmap fil indicate cluster volume allocate file free boot fil contain startup code windows locate particular disk address find easily simple rom bootstrap loader boot file contain physical address bad cluster file keep track bad area volume ntfs use record error recovery keep ntfs metadata actual file useful property dis- cuss section b.3.3.6 cache manager cache file datum ntfs metadata reside file datum cache mechanism ordinary datum simple file system power failure wrong time damage file system datum structure severely entire volume scramble unix file system include ufs zf store redundant metadata disk recover crash fsck program check file system datum structure restore forcibly consistent state restore involve delete damage file free data cluster write user datum properly record file system metadata structure checking slow process cause loss significant amount datum ntfs take different approach file system robustness ntfs file- system data structure update perform inside transaction data structure alter transaction write log record contain redo undo information datum structure change transaction write commit record log signify transaction succeed crash system

transaction write commit record log signify transaction succeed crash system restore file system datum structure commit successfully crash periodically usually 5 second checkpoint record write log system need log record prior checkpoint recover crash discard log file grow bound time system startup ntfs volume access ntfs automatically perform scheme guarantee user file content correct crash ensure file system datum structure metadata file undamaged reflect consistent state exist prior crash possible extend transaction scheme cover user file microsoft take step windows vista log store metadata file beginning volume create fix maximum size file system format section logging area circular queue log record restart area hold context information position logging area ntfs start read recovery fact restart area hold copy information recovery possible copy damage crash logging functionality provide log file service addition write log record perform recovery action log file service keep track free space log file free space get low log- file service queue pende transaction ntfs halt new o operation progress operation complete ntfs call cache manager flush datum reset log file perform queue transaction security ntfs volume derive windows object model ntfs file reference security descriptor specify owner file access control list contain access permission grant deny user group list early version

ntfs separate security descriptor attribute file begin windows 2000 security descriptor attribute point share copy significant saving disk caching space file identical security normal operation ntfs enforce permission traversal directory file path name compatibility posix check enable traversal check inherently expensive modern parsing file path name use prefix matching directory- directory parsing path name prefix matching algorithm look string cache find entry long match example entry foobardir match foobardir2dir3myfile prefix
match cache allow path traversal begin deeply tree save step enforce traversal check mean

user access check directory level instance user lack permission traverse foobar start access foobardir error volume management fault tolerance ftdisk fault tolerant disk driver windows instal pro- vide way combine multiple disk drive logical volume improve performance capacity reliability disk 1 2.5 gb disk 2 2.5 gb disk c fat 2 gb logical drive d ntfs 3 gb volume set drive volume set raid set way combine multiple disk concatenate logically form large logical volume show figure b.7 windows logical volume call volume set consist 32 physical partition volume set contain ntfs volume extend disturbance datum store file system bitmap metadata ntfs volume simply extend cover newly add space ntfs continue use lcn mechanism use single physical disk ftdisk driver supply mapping logical volume offset offset way combine multiple physical partition interleave block round robin fashion form stripe set scheme call raid level 0 disk striping raid redundant array inexpen- sive disk section 11.8 ftdisk use stripe size 64 kb 64 kb logical volume store physical partition second 64 kb second physical partition partition contribute 64 kb space allocation wrap disk allocate second 64 kb block stripe set form large logical volume physical layout improve o bandwidth large o disk transfer datum parallel windows support raid level 5 stripe set parity raid level 1 mirroring sector

level 5 stripe set parity raid level 1 mirroring sector sparing cluster remapping deal disk sector bad ftdisk use hardware technique call sector sparing ntfs use software technique call cluster remapping sector sparing hardware capability provide disk drive disk drive format create map logical block number good sector disk leave extra sector unmapped spare sector fail ftdisk instruct disk drive substitute spare cluster remapping software technique perform file system disk block go bad ntfs substitute different unallocated block change affected pointer mft ntfs make note bad block allocate file disk block go bad usual outcome data loss sector sparing cluster remapping combine fault tolerant volume mask failure disk block read fail system reconstruct miss datum read mirror calculate exclusive parity stripe set parity reconstruct datum store new location obtain sector sparing cluster remapping ntfs perform datum compression individual file datum file directory compress file ntfs divide file datum compression unit block 16 contiguous cluster compression unit write data compression algorithm apply result fit few 16 cluster compress version store read ntfs determine datum compress length store compression unit 16 cluster improve per- formance read contiguous compression unit ntfs prefetche decompress ahead application request sparse file file contain zero

ntfs use tech- nique save space cluster contain zero write actually allocate store disk instead gap leave sequence virtual cluster number store mft entry file read file ntfs find gap virtual cluster number zero fill portion caller buffer technique unix mount points symbolic links hard links mount point form symbolic link specific directory ntfs introduce windows 2000 provide mechanism organize disk volume flexible use global name like drive letter mount point implement symbolic link associate datum contain true volume ultimately mount point sup- plant drive letter completely long transition dependence application drive letter scheme windows vista introduce support general form symbolic link similar find unix link absolute

relative point object exist point file directory volume ntfs support hard link single file entry directory volume ntfs keep journal describe change file system user mode service receive notification change journal identify file change read journal search indexer service use change journal identify file need index file replication service use identify file need replicate network volume shadow copies windows implement capability bring volume know state create shadow copy consistent view volume technique know snapshot file system make shadow copy volume form copy write block modify shadow copy create store original form copy achieve consistent state volume require cooperation application system know datum application stable state application server version windows use shadow copy efficiently maintain old version file store file server allow user document store file server exist early point time user use feature recover file accidentally delete simply look previous version file pull backup medium windows support peer peer client server networking facility network management networking component win- dows provide datum transport interprocess communication file sharing network ability send print job remote printer describe networking windows mention internal networking interface network device interface specificatio ndis transport driver interface tdi ndis interface develop 1989 microsoft 3com separate network adapter transport protocol change affect ndis reside interface data link network layer iso model enable protocol operate different network adapter term iso model tdi interface transport layer layer 4 session layer layer 5 interface enable session layer component use available transport mechanism similar reasoning lead stream mechanism unix tdi support connection base connectionless transport function send type datum windows implement transport protocol driver driver load unload system dynamically practice system typically reboot change windows come networking protocol discuss number protocol server message block smb protocol introduce ms dos 3.1 system use protocol send o request network smb protocol message type session control message

o request network smb protocol message type session control message command start end redirector connection share resource server redirector use file message access file server printer message send datum remote print queue receive status information queue message message communicate workstation version smb protocol publish common internet fil system cifs support number operating system transmission control protocol internet protocol transmission control protocol internet protocol tcp ip suite internet de facto standard networking infrastructure windows use tcp ip connect wide variety operating system hardware platform windows tcp ip package include simple network management protocol snm dynamic host configuration proto- col dhcp old windows internet service wins windows vista introduce new implementation tcp ip support ipv4 ipv6 network stack new implementation support offload network stack advanced hardware achieve high performance server windows provide software firewall limit tcp port program network communication network firewall com- monly implement router important security measure have firewall build operate system make hardware router unnecessary provide integrated management easy use point point tunneling protocol point point tunneling protocol pptp protocol provide win- dows communicate remote access server module run win- dows server machine client system connect internet remote access server encrypt datum send connec- tion support multiprotocol virtual private network vpns http protocol information world wide web windows implement http kernel mode driver web server operate low overhead connection networking stack http fairly general protocol windows make available transport option web distribute authoring versioning protocol web distribute authoring versioning webdav http base proto- col collaborative authoring network windows build webdav redirector file system build directly file system enable webdav work file system feature encryption personal file store securely public place webdav use http protocol windows cache file locally pro- grams use read write operation part file name pipe connection orient messaging mechanism process use name pipe communicate process machine

orient messaging mechanism process use name pipe communicate process machine name pipe access file system interface security mechanism file object apply name pipe smb protocol support name pipe communication process different system format pipe name follow uniform naming convention unc unc look like typical remote file format server nameshare namexyz server identify server network share identify resource available network user directory file name pipe printer xyz normal file path remote procedure call remote procedure rpc client server mechanism enable application machine procedure code machine client call local procedure stub routine pack argument message send network particular server process client stub routine block server unpack message call procedure pack return result mes- sage send client stub client stub unblock receive message unpack result rpc return caller packing argument call marshaling client stub code descriptor necessary pack unpack argument rpc compile specification write microsoft interface definitio windows rpc mechanism follow widely distributed- computing environment standard rpc message program write use windows rpcs highly portable rpc standard detail hide architectural difference computer size binary number order byte bit computer word specify standard data format rpc message component object model component object model com mechanism interprocess commu- nication develop windows com object provide define interface manipulate datum object instance com infras- tructure microsoft object linking embedding ole technology insert spreadsheet microsoft word document windows service provide com interface windows distribute extension call dcom network utilize rpc provide transparent method develop distribute application redirector server windows application use windows o api access file remote computer local provide remote com- puter run cifs server provide windows aredirector client object forward o request remote system satisfy server performance security redirector server run kernel mode detail access remote file occur follow 1 application call o manager request file open file standard unc format 2 o manager build o

open file standard unc format 2 o manager build o request packet describe section 3 o manager recognize access remote file call driver call multiple universal naming convention provider mup 4 mup send o request packet asynchronously register 5 redirector satisfy request respond mup avoid ask redirector question future mup use cache remember redirector handle file 6 redirector send network request remote system 7 remote system network driver receive request pass 8 server driver hand request proper local file system driver 9 proper device driver call access datum 10 result return server driver send datum request redirector redirector return datum call application o manager asimilar process occur application use win32 network api unc service module call multi provider router instead mup portability redirector server use tdi api network trans- port request express high level protocol default smb protocol describe section b.6.2 list redirector maintain system hive registry distributed file system unc name convenient multiple file server available serve content unc name explicitly include server windows support distributed file syste dfs protocol allow network administrator serve file multiple server single distribute space folder redirection client caching improve pc experience user frequently switch com- puter windows allow administrator user roam profile user preference setting server folder redirection automatically store user document file server work computer long attach network user take laptop airplane user line access redirect file windows use client caching csc csc computer line copy server file local machine well performance file push server change computer disconnected file available update server defer time computer online networked environment natural group user student computer laboratory school employee department busi- ness frequently want member group able access share resource computer group manage global access right group windows use concept domain pre- viously domain relationship whatsoever domain system dns map internet host name ip address

whatsoever domain system dns map internet host name ip address closely relate specifically windows domain group windows workstation server share common security policy

user database windows use kerberos protocol trust authentication windows domain thing kerberos realm windows use hierarchical approach establish trust relationship related domain trust rela- tionship base dns allow transitive trust flow hierarchy approach reduce number trust require n domain n (n 1 o(n workstation domain trust domain controller correct information access right user load user access token lsass user retain ability restrict access workstation matter domain controller active directory windows implementation lightweight directory- access protocol ldap service active directory store topology infor- mation domain keep domain base user group account password provide domain base store windows feature need windows group policy administrator use group policy establish uniform standard desktop preference software corporate information technology group uniformity drastically reduce cost computing win32 api fundamental interface capability windows section describe main aspect win32 api access kernel object sharing object process process management interprocess com- munication memory management access kernel object windows kernel provide service application program use application program obtain service manipulate kernel object process gain access kernel object name xxx call createxxx function open handle instance xxx handle unique process depend object open create function fail return 0 return special constant name invalid handle value process close handle call closehandle function system delete object count handle reference object process drop zero share object process windows provide way share object process way child process inherit handle object parent call createxxx function parent supply security attribute structure binherithandle field set true field create inheritable handle child process create pass value true createprocess function binherithandle argument figure b.8 show code sample create semaphore handle inherit child security attribute sa sa.nlength = sizeof(sa sa.lpsecuritydescriptor = null sa.binherithandle = true handle semaphore = createsemaphore(&sa 1 1 null char comand line[132 ostrstream ostring(command line sizeof(command line ostring < < semaphore < < ends createprocess("another process.exe

command line null null true code enable child share object inherit handle process handle semaphore = createsemaphore(null 1 1 mysem1 process b handle b semaphore = opensemaphore(semaphore access code share object lookup assume child process know handle share parent child achieve interprocess communication share object example figure b.8 child process get value handle command line argument share semaphore parent process second way share object process object object create second process open method drawback windows provide way check object choose exist object space global regard object type instance application create share single object name foo distinct object possibly different type desire name object advantage unrelated process readily share process call createxxx function supply parameter second process get handle share object call openxxx createxxx show example figure b.9 way share object duplicatehandle function method require method interprocess communication pass duplicated handle give handle process value handle process second process handle object share example method show figure b.10 windows process load instance application thread executable unit code schedule kernel dispatcher process contain thread process create thread process call createprocess api routine load dynamic link library process create initial thread process additional thread create createthread function thread create stack default 1 mb specify argument createthread process want process b access semaphore process handle semaphore = createsemaphore(null 1 1 null send value semaphore process b message share memory object process b handle process = openprocess(process access false process d handle b semaphore duplicatehandle(process semaphore getcurrentprocess b semaphore 0 false duplicate access use b semaphore access semaphore code share object pass handle priority win32 environment base native kernel nt scheduling model priority value choose win32 api use priority class 1 idle priority class nt priority level 4 2 normal priority class nt priority level 8) 3 high priority class nt priority level 13 4

realtime priority class nt priority level 24 process typically member normal priority class parent process idle priority class class specify createprocess call priority class process default thread execute process change setpriorityclass function pass argument start command user increase scheduling priority privilege process realtime priority class administrator power user privilege default user run interactive process system need schedule process thread provide good responsiveness reason windows special scheduling rule process nor- mal priority class windows distinguish process associate foreground window screen background process process move foreground windows increase scheduling quantum thread factor 3 cpu bind thread foreground process run time long similar thread thread start initial priority determine class priority alter setthreadpriority function function take argument specify priority relative base priority class thread priority lowest base 2 thread priority normal base 1 thread priority normal base + 0 thread priority normal base + 1 thread priority highest base + 2 designation adjust priority recall section b.3.2.2 kernel priority class 16–31 real- time class 1–15 variable class thread priority idle set priority 16 real time thread 1 variable priority thread thread priority time critical set priority 31 real time thread 15 variable priority thread discuss section b.3.2.2 kernel adjust priority variable class thread dynamically depend thread o bind cpu bind win32 api provide method disable adjustment setprocesspriorityboost setthreadpriorityboost function thread suspend resume thread create suspend state place suspend state later use suspendthread function suspend thread schedule kernel dispatcher move sus- pende state use resumethread function function set counter thread suspend twice resume twice run synchronize concurrent access share object thread kernel pro- vide synchronization object semaphore mutexe dispatcher object discuss section b.3.2.2 thread synchronize kernel service operate kernel object thread process file dispatcher object synchronization ker- nel dispatcher object achieve use waitforsingleobject waitformultipleobjects function function wait dispatcher object signal method synchronization available thread

function function wait dispatcher object signal method synchronization available thread process want execute code exclusively win32 critical section object user mode mutex object acquire release enter kernel multiprocessor win32 critical section attempt spin wait critical section hold thread release spinning take long acquire thread allocate kernel mutex yield cpu critical section particularly efficient kernel mutex allocate contention attempt spin mutexe program actually contend saving significant critical section thread process initializecriticalsection thread want acquire mutex call entercriticalsection later call leavecritical- section release mutex tryentercriticalsection function attempt acquire mutex block program want user mode reader writer lock mutex win32 support slim reader writer srw lock srw lock api similar critical section initializesrwlock exclusive share depend thread want write access read access object protect lock win32 api support condition variable critical section srw lock repeatedly create delete thread expensive application service perform small amount work instantiation win32 thread pool provide user mode program service queue work request submit submitthreadpoolwork function api bind callback waitable handle regis- terwaitforsingleobject api work timer createthread- pooltimer waitforthreadpooltimercallbacks bind call- back o completion queue bindiocompletioncallback goal thread pool increase performance reduce mem- ory footprint thread relatively expensive processor execute thread time matter thread available thread pool attempt reduce number runnable thread slightly delay work request reuse thread request provid- e thread effectively utilize machine cpu wait o- timer callback api allow thread pool reduce number thread process far few thread necessary process devote separate thread service waitable handle timer completion port fibe user mode code schedule accord

user define scheduling algorithm fiber completely user mode facility kernel aware exist fiber mechanism use windows thread cpu execute fiber fiber cooperatively schedule mean preempt explicitly yield thread run fiber yield thread fiber schedule run time system programming language run time code system create fiber call convertthreadtofiber createfiber primary difference function createfiber begin execute fiber create begin execution application switchtofiber application terminate fiber call deletefiber fiber recommend thread use win32 api standard c library function potential incompatibility win32 user- mode thread thread environment block teb contain numerous thread field win32 api fiber share teb thread run lead problem win32 interface put state information teb fiber information overwrite different fiber fiber include win32 api facilitate porting legacy unix application write user mode thread model pthread user mode scheduling ums concrt new mechanism windows 7 user mode scheduling ums address limitation fiber recall fiber unreliable execute win32 api teb thread run fiber block kernel user scheduler lose control cpu time kernel dispatcher take scheduling problem result fiber change kernel state thread priority impersonation token start asynchronous o.

ums provide alternative model recognize windows thread actually thread kernel thread kt user thread ut type thread stack set save register kt ut appear single thread programmer ut block enter kernel implicit switch correspond kt take place ums use ut teb uniquely identify ut ut enter kernel explicit switch kt correspond ut identify current teb reason kernel know ut run ut invoke user mode scheduler fiber ums scheduler switch uts include switch ut enter kernel kt block happen kernel switch scheduling thread ums call primary use thread reenter user mode scheduler pick ut run eventually block kt complete operation ready return user mode ums reentere user mode scheduler run different ut ums queue ut correspond complete kt completion list user mode user mode scheduler choose new ut switch examine completion list treat ut list candidate scheduling unlike fiber ums intend directly program- mer detail write user mode scheduler challenging ums include scheduler scheduler come pro- gramming language library build ums microsoft visual studio 2010 ship concurrency runtime concrt concurrent programming framework c++ concrt provide user mode scheduler facility decompose program task schedule available cpu concrt provide support par style con- primary thread run user mode trap code switch park kt kt block = primary return user mode kt unblock park = queue ut completion ut completion list struct rudimentary resource management task synchronization primitive key feature ums depict figure b.11 winsock windows socket api winsock session layer interface largely compatible unix socket add windows extension provide standardized interface transport protocol different addressing scheme winsock application run winsock compliant protocol stack winsock undergo major update windows vista add tracing ipv6 support impersonation new security api feature winsock follow windows open system architecture wosa model provide standard service provider interface spi applica- tion networking protocol application load unload layered protocol build additional

tion networking protocol application load unload layered protocol build additional functionality additional security transport protocol layer winsock support asynchronous opera- tion notification reliable multicasting secure socket kernel mode socket support simple usage model like wsaconnect- byname function accept target string specify ip address server service port number destination port ipc windows messaging win32 application handle interprocess communication way way share kernel object windows messaging facility approach particularly popular win32 gui application thread send message thread window call postmessage postthreadmessage sendmessage sendthreadmessage sendmessagecallback post message send message differ way post routine asynchronous return immediately call thread know message allocate 16 mb address space void buf = virtualalloc(0 0x1000000 mem reserve | mem commit upper 8 mb allocate space virtualalloc(buf + 0x800000 0x800000 mem commit page readwrite memory decommit memory virtualfree(buf + 0x800000 0x800000 mem decommit release allocate address space virtualfree(buf 0 mem release code fragment allocate virtual memory actually deliver send routine synchronous block caller message deliver process addition send message thread send datum mes- sage process separate address space datum copy system copy datum call sendmessage send message type wm copydata copydatastruct datum structure contain length address datum transfer message send windows copy datum new block memory give virtual address new block receive process win32 thread input queue receive mes- sage win32 application getmessage handle event input queue queue fill second system mark application respond win32

api provide way application use memory virtual memory memory map file heap thread local storage application call virtualalloc reserve commit virtual memory virtualfree decommit release memory function enable application specify virtual address memory allocate operate multiple memory page size example function appear figure b.12 process lock commit page physical memory call virtuallock maximum number page process lock 30 process call setprocessworkingsetsize increase maximum work set size way application use memory memory map file address space memory mapping convenient way open file create exist handle hfile = createfile("somefile generic read | generic write file share read | file share write null open file attribute normal null create file mapping 8 mb size handle hmap = createfilemapping(hfile page readwrite sec commit 0 0x800000 shm 1 view space map void buf = mapviewoffile(hmap file map access 0 0 0 0x800000 map file unmap file code fragment memory mapping file process share memory process map file virtual memory memory mapping multistage process example figure b.13 process want map address space share memory region process file need process call createfilemap- ping file handle 0xffffffff particular size result file map object share inheritance lookup handle heap provide way application use memory mal- loc free standard c. heap win32 environment region reserved address space win32 process initialize create default heap win32 application multithreaded access heap synchronize protect heap space allocation data structure damage concurrent update multiple thread win32 provide heap management function process allocate manage private heap function heapcreate hea- palloc heaprealloc heapsize heapfree heapdestroy win32 api provide heaplock heapunlock function enable thread gain exclusive access heap unlike virtuallock function perform synchronization lock page original win32 heap optimize efficient use space lead significant problem fragmentation address space large server program run long period time new low fragmentation heap lfh design introduce windows xp

greatly reduce fragmen- reserve slot variable dword var index = t1salloc set value 10 t1ssetvalue(var index 10 value int var t1sgetvalue(var index release index code dynamic thread local storage tation problem windows 7 heap manager automatically turn lfh fourth way application use memory thread local storage tls mechanism function rely global static datum typically fail work properly multithreaded environment instance c run time function strtok use static variable track current position parse string concurrent thread execute strtok cor- rectly need separate current position variable tls provide way maintain instance variable global function execute share thread tls provide dynamic static method create thread local storage dynamic method illustrate figure b.14 tls mechanism allocate global heap storage attach thread environment block windows allocate user mode thread teb readily accessible thread tls thread state information user mode use thread local static variable application declare variable follow ensure thread private copy declspec(thread dword cur pos = 0

microsoft design windows extensible portable operating system able advantage new technique hardware windows support multiple operating environment symmetric multiprocessing include 32 bit 64 bit processor numa computer use kernel object provide basic service support client server computing enable windows support wide variety application environ- ment windows provide virtual memory integrated caching preemptive scheduling support elaborate security mechanism include interna- tionalization feature windows run wide variety computer user choose upgrade hardware match budget performance requirement need alter application run type operate system windows describe major list design goal windows describe detail describe booting process windows system describe main architectural layer windows kernel job object manager type service process manager provide local procedure responsibility o manager type networking windows support windows implement transport protocol describe networking protocol ntfs namespace organize ntfs handle data structure ntfs recover system crash guarantee recovery take place windows allocate user memory describe way application use memory win32 api russinovich et al 2017 provide overview windows 7 consider- able technical detail system internal component brown 2000 present detail security architecture windows microsoft developer network library http://msdn.microsoft.com supply wealth information windows microsoft product include documentation publish api iseminger 2000 provide good reference windows active direc- tory detailed discussion write program use win32 api appear richter 1997 source code 2005 wrk version windows kernel collection slide crk curriculum material available www.microsoft.com/windowsacademic use university k. brown programming windows security addison wesley d. iseminger active directory services microsoft windows 2000 technical reference microsoft press 2000 j. richter advanced windows microsoft press 1997 russinovich et al 2017 m. russinovich d. a. solomon a. ionescu win- dows internals 1 seventh edition

microsoft press 2017 chapter firs write 1991 update time chapter 20 present depth examination linux operate system chapter examine popular unix version unixbsd start present brief history unix operate system describe system user programmer interface finally discuss internal data structure algorithm freebsd kernel support user programmer interface version unix develop 1969 ken thompson research group bell laboratories use idle pdp-7 thomp- son soon join dennis ritchie member research group produce early version unix ritchie previously work multics project multics strong influence new operate system unix pun multics basic organization file system idea command interpreter shell user process use separate process command original line edit character erase character erase entire line numerous feature come directly multics idea operate system mit ctss xds-940 system ritchie thompson work quietly unix year move pdp-11/20 second version version rewrite operate system system program language c instead previously assembly language c develop bell laborato- ries support unix unix move large pdp-11 model 11/45 11/70 multiprogramming enhancement add rewrite c move system 11/45 hardware support multiprogramming unix develop widely bell laboratories gradually spread university version widely available out- bell laboratories version 6 release 1976 version number early unix system correspond edition number unix program- mer manual current distribution code manual revise independently 1978 version 7 distribute unix system run pdp-11/70 interdata 8/32 ancestor modern unix system particular soon port pdp-11 model vax com- puter line version available vax know 32v. research continue unix support group distribution version 7 1978 unix support group usg assume administrative control responsibility

research group distribution unix at&t parent organization bell labora- tories unix product simply research tool research group continue develop version unix support internal computing version 8 include facility call stream o system allow flexible configuration kernel ipc module contain rfs remote file system similar sun nfs current version version 10 release 1989 available bell laboratories usg mainly provide support unix at&t. external distribution usg system iii 1982 system iii incorporate feature version 7 32v feature unix system develop group research example feature unix rt real time unix system numerous portion programmer work bench pwb software tool package include system iii usg release system v 1983 largely derive system iii divestiture bell operate company at&t leave at&t position market system v aggressively usg restructure unix system development laboratory usdl release unix system v release 2 v.2 1984 unix system v release 2 version 4 v.2.4 add new implementation virtual memory copy write paging share memory usdl turn replace at&t information systems attis distribute system v release 3 v.3 1987 v.3 adapt v8 imple- mentation stream o system make available streams include rfs nfs like remote file system mention early berkeley begin development small size modularity clean design early unix system lead unix- base work numerous computer science organization rand bbn university illinois harvard purdue dec influential unix development group outside bell laboratories at&t university california berkeley bill joy ozalp babaoglu berkeley vax unix work 1978 add virtual memory demand paging page replacement 32v produce 3bsd unix version implement facility unix system large virtual memory space 3bsd allow development large program berkeley franz lisp memory management work convince defense

advanced research projects agency darpa fund berkeley development standard unix system government use 4bsd unix result 4 bsd work darpa guide steering committee include notable people unix networking community goal project provide support darpa inter- net networking protocol tcp ip support provide general manner possible 4.2 bsd communicate uniformly diverse network facility include local area network ethernets token ring wide area network nsfnet implementation important reason current popularity protocol ven- dor unix computer system basis implementation operate system permit internet grow 60 connect network 1984 8,000 network estimate 10 million user 1993 addition berkeley adapt feature contemporary operating system improve design implementation unix terminal line editing function tenex tops-20 operate system provide new terminal driver new user interface c shell new text editor ex vi compiler pascal lisp new system program write berkeley 4.2 bsd certain efficiency improvement inspire vms operate system unix software berkeley release berkeley software distribu- tion bsd convenient refer berkeley vax unix system follow 3 bsd 4 bsd actually specific release notably 4.1 bsd 4.2 bsd 4.2 bsd distribute 1983 culmination original berkeley darpa unix project equivalent version pdp-11 system 2.9 bsd 1986 4.3 bsd release similar 4.2 bsd include numerous internal change bug fix performance improvement new facility add include support xerox network version 4.3 bsd tahoe release 1988 include improved networking congestion control tcp ip performance disk configuration separate device driver read disk expanded time zone support include 4.3 bsd tahoe actually develop cci tahoe system computer console inc.

power 6 computer usual vax base corresponding pdp-11 release 2.10.1bsd distribute usenix association publish 4.3 bsd manual 4.3.2 bsd reno release see inclusion implementation iso osi networking berkeley release 4.4 bsd finalize june 1993 include new x.25 networking support posix standard compliance radically new file system organization new virtual file system interface support stackable file system allow file system layer easy inclusion new feature implementation nfs include release section 15.8 new log base file system chapter 11 4.4 bsd virtual memory system derive mach describe section a.13 change enhanced security improved kernel structure include release version 4.4 berkeley halt research effort spread unix unix 4 bsd operate system choice vax initial release 1979 release ultrix dec bsd implementation 4 bsd good choice research networking installation current set unix operate system limit bell laboratories currently own lucent technology berkeley sun microsystems help popularize bsd flavor unix ship- pe sun workstation unix grow popularity move computer computer system wide variety unix unix- like operate system create dec support unix ultrix workstation replace ultrix unix derive operate system osf/1 microsoft rewrite unix intel 8088 family call xenix windows nt operate system heavily influence unix available general purpose computer run personal computer workstation minicomputer mainframe supercomputer apple macintosh iis cray iis wide availability environment range academic military manufacture process control system base version 7 system iii 4.2 bsd wide popularity unix computer vendor unix portable operate system user expect unix environment independent specific computer manufacturer large number implementation system lead remarkable variation program- ming user interface distribute vendor true vendor indepen- dence application program developer need consistent interface inter- face allow unix application run unix system certainly current situation issue important unix preferred program development platform application range database graphic networking lead strong market demand

unix standard standardization project undertake /usr group 1984 standard sponsor uniforum industry user group official standard body continue effort include ieee iso posix standard x open group international consor- tium complete xpg3 common application environment subsume ieee interface standard unfortunately xpg3 base draft ansi c standard final specification need redone xpg4 1989 ansi standard body standardize c program- ming language produce ansi c specification vendor quick project continue flavor unix converge lead programming interface unix allow unix popular fact separate set powerful unix vendor work problem at&t guide unix international ui open software foundation osf agree follow posix standard recently vendor involve group agree standardization cose agreement history unix version 1993 at&t replace attis group 1989 unix software organization uso ship merge unix system v release 4 system combine feature system v 4.3 bsd sun sunos include long file name berkeley file system virtual memory management symbolic link multiple access group job control reliable signal conform publish posix standard posix.1 uso produce svr4 independent at&t subsidiary name unix system laboratories usl 1993 purchase novell inc. figure c.1 summarize relationship version unix unix system grow personal project bell labora- tories employee operate system define multinational standard- ization body time unix excellent vehicle academic study believe remain important operating system theory practice example tunis operate system xinu oper- ating system minix operate system base concept unix develop explicitly classroom study plethora ongoing unix relate research system include mach chorus comandos roisin original developer ritchie thompson honor 1983 association computing machinery turing award work history freebsd specific unix version chapter intel version freebsd system implement interesting operating system concept demand paging clustering networking freebsd project begin early 1993

produce snapshot 386 bsd solve problem resolve exist patch mechanism 386 bsd derive 4.3 bsd lite net/2 release june 1992 william jolitz freebsd coin david greenman 1.0 release december 1993 freebsd 1.1 release 1994 version base 4.3 bsd lite legal issue ucb novell require 4.3 bsd lite code long final 4.3 bsd lite release july 1994 freebsd 1.1.5.1 freebsd reinvent base 4.4bsd lite code incom- plete freebsd 2.0 release november 1994 later release include 2.0.5 june 1995 2.1.5 august 1996 2.1.7.1 february 1997 2.2.1 april 1997 2.2.8 november 1998 3.0 october 1998 3.1 february 1999 3.2 1999 3.3 september 1999 3.4 december 1999 3.5 june 2000 4.0 march 2000 4.1 july 2000 4.2 november 2000 goal freebsd project provide software purpose string attach idea code wide possible use provide benefit present run primarily intel platform alpha platform support work underway port processor platform unix design time share system standard user interface shell simple replace desire file system multilevel tree allow user create subdirectory user datum file simply sequence byte disk file o device treat similarly possible device dependency peculiarity keep kernel possible kernel confine device driver unix support multiple process process easily create new pro- cesse cpu scheduling simple priority algorithm freebsd use demand paging mechanism support memory management cpu scheduling decision swapping system suffer excess paging unix originate thompson ritchie system convenience small understand algo- rithm select simplicity speed sophistication intent kernel library

provide small set facility suf- ficiently powerful allow person build complex system need unix clean design result imitation modification designer unix significant knowledge operating system unix elaborate design spell implementation flexibility appear key factor development system design principle involve explicit outset unix system design programmer programmer interactive facility program development high priority facility include program check collection source file program need compile compiling source code control system sccs successive version file available have store entire content step primary version control system unix concurrent versions system cvs large number developer operate code operate system write c develop support unix thompson ritchie enjoy program assembly language avoidance assembly language necessary uncertainty machine unix run greatly simplify problem move unix hardware system beginning unix development system unix source available online developer system development primary system pattern development greatly facilitate discovery deficiency fix new possibility implementation encourage plethora unix variant exist today benefit outweigh disadvantage break fix local site need wait release system fix new facility incorporate later distribution size constraint pdp-11 early computer unix force certain elegance system elaborate algorithm deal pathological condition unix control crash call panic instead attempt cure condition unix try pre- vent system use brute force macro expansion unix develop subtle simple approach early strength unix produce popularity turn produce new demand challenge strength unix task networking graphic real time operation fit original text orient model change certain internal facility new programming interface add support new facility particularly window interface require large amount code radically increase size system instance networking windowing double size system pattern turn point continued strength unix new development occur industry

unix usually absorb like operate system unix consist separable part kernel system program view unix operating system layer show figure c.2 system interface physical hardware kernel kernel provide file system cpu scheduling memory management operating system function system call system program use kernel support system call provide useful function compilation file manipulation system call define programmer interface unix set system program commonly available define user interface programmer user interface define context kernel support system program write c unix programmer manual present system call c function asystem program write c freebsd pentium generally move alpha freebsd system simply recompile system different detail system call know compiler feature major reason portability unix program system call unix roughly group category file manipulation process control information manipulation chapter 2 list fourth category device manipulation device unix treat special file system call support file device extra system set device parameter shell command compiler interpreter system interface kernel kernel interface hardware swap block o disk tape driver character o system disk tape 4.4bsd layer structure file unix sequence byte different program expect level structure kernel impose structure file instance convention text file line ascii character separate single newline character linefeed character ascii kernel know convention file organize tree structured directory directory them- selves file contain information find file path file text string identify file specify path directory structure file syntactically consist individual file element separate slash character example /usr local font slash indicate root directory tree call root directory element usr subdirectory root local subdirectory usr font file directory directory local font ordinary file directory determine path syntax unix file system absolute path name relative path name absolute path name start root file system dis- tinguishe slash beginning path /usr local font absolute path relative path name start current directory attribute process access path local font

name start current directory attribute process access path local font indicate file directory name font directory local current directory /usr file know directory multiple name know link link treat equally operate system freebsd support symbolic link file contain path file kind link know hard link soft link soft symbolic link unlike hard link point directory cross file system boundary file directory hard link directory file hard link parent directory current directory /user jlp program /bin wdf refer /user jlp bin wdf hardware device name file system device special file special file know kernel device interface nonetheless access user system call figure c.3 show typical unix file system root normally contain small number directory /kernel binary boot image operate system /dev contain device special file /dev console /dev lp0 /dev mt0 /bin contain binary essential unix system program binary /usr bin application system program text formatter /usr compat program operate system linux /usr local bin system program write local site library file c pascal fortran subroutine library keep /lib /usr lib /usr local lib file user store separate directory user typically /usr user directory carol normally /usr carol large system directory group ease administration create file structure /usr prof avi /usr staff carol administrative file program password file keep /etc typical unix directory structure temporary file /tmp normally erase system boot /usr tmp directory considerably structure exam- ple font description table troff formatter merganthaler 202 typesetter keep /usr lib troff dev202 convention concern location specific file directory define programmer program operating system kernel need /etc

init initialize terminal process operable system call basic file manipulation creat open read write close unlink trunc creat system give path create file truncate exist exist file open open system take path mode read write read write return small integer call file descriptor file descriptor pass read write system buffer address number byte transfer perform datum transfer file afile close file descriptor pass close system trunc reduce length file 0 file descriptor index small table open file process descriptor start 0

seldom high 6 7 typical program depend maximum number simultaneously open file read write update current offset file associate file table entry determine position file read write lseek system allow position reset explicitly allow creation sparse file file hole dup dup2 system call produce new file descriptor copy exist fcntl system addition examine set parameter open file example succeed write open file append end file additional system ioctl manipulate device parameter set baud rate serial port rewind tape instance information file size protection mode owner obtain stat system system call allow information change rename change file chmod change protection mode chown change owner group system call variant apply file descriptor instead file name link system make hard link exist file create new exist file link remove unlink system link file delete symlink system make symbolic link directory mkdir system delete rmdir current directory change cd standard file call open directo- rie inadvisable directory internal structure preserve instead set system call provide open directory step file entry directory close directory perform function opendir readdir closedir process program execution process identify process identifier integer new process create fork system new process consist copy address space original process program variable value process parent child continue execution instruction fork difference return code fork zero new child process nonzero process identifier child return parent typically execve system fork process replace process virtual memory space new program execve system load binary file memory destroy memory image program contain execve system start process terminate exit system parent process wait event wait system child process crash system simulate exit wait system provide process id terminate child parent tell possibly child terminate second system wait3 similar wait allow parent collect performance statistic child time child exit time parent complete wait system call child defunct defunct process exist merely parent

wait system call child defunct defunct process exist merely parent collect status information parent process defunct process exit child defunct process inherit init process turn wait zombie process typical use facility show figure c.4 simple form communication process pipe apipe create fork endpoint set fork execve pipe essentially queue byte process pipe access file descriptor like ordinary file process write pipe read pipe size original pipe system fix system freebsd pipe implement socket system variable sized buffer read pipe write pipe cause process block state pipe change special arrangement need pipe place parent child read write user process descendant original process call init process identifier 1 terminal port available interactive use getty process fork init getty process initialize termi- nal line parameter wait user login pass shell fork subprocess execute program execve argument login process login process collect user password encrypt compare result encrypt string take file /etc passwd comparison successful user allow log login process execute shell command interpreter set numeric user identifier process user log shell user identifier find /etc passwd user login shell user ordinarily communicate rest login session shell fork subprocesse command user tell execute user identifier kernel determine user permission certain system call especially involve file access group identifier provide similar privilege collection user freebsd process group simultaneously login process put shell group permit user file /etc passwd /etc group user identifier kernel effective user identifier real user identifier effective user identifier determine file access permission file program load execve setuid bit set inode effective user identifier process set user identifier owner file real user identifier leave scheme allow certain process ordinary privilege executable ordinary user setuid idea patent dennis ritchie u.s. patent 4,135,240

distinctive feature unix similar setgid bit exist group process determine real effective user identifier getuid geteuid call respectively getgid getegid call determine process real effective group identifier respectively rest process group find getgroups system signal facility handle exceptional condition similar software interrupt 20 different signal correspond distinct condition signal generate keyboard interrupt error process bad memory reference number asynchronous event timer job control signal shell signal generate kill system interrupt signal sigint stop command com- mand complete usually produce ˆc character ascii 3 4.2bsd important keyboard character define table termi- nal redefine easily quit signal sigquit usually produce ˆbs character ascii 28 quit signal stop currently execut- e program dump current memory image file name core current directory core file debugger sigill produce illegal instruction sigsegv attempt address memory outside legal virtual memory space process arrangement signal ignore effect routine user process signal handler call signal handler safely thing return catch signal exit system modify global variable signal kill signal number 9 sigkill ignore catch signal handler sigkill example kill runaway process ignore signal sigint sigquit signal lose signal kind send previous signal accept process direct signal overwrite signal see process word signal handler tell process occurrence signal relative priority unix signal different signal send process time know process receive signal originally intend deal exceptional event true unix feature signal use steadily expand 4.1bsd introduce job control use signal start stop subprocesse demand facility allow shell control multiple process start stop background user wish sigwinch signal invent sun microsystems inform process window output display change size signal deliver urgent datum network connection user want reliable signal bug fix inherent race condi- tion old signal implementation 4.2bsd bring race free reliable separately implement signal capability allow individual signal block critical section new system let pro- cess sleep interrupt similar hardware interrupt

new system let pro- cess sleep interrupt similar hardware interrupt functionality capability posix standard group related process frequently cooperate accomplish common task instance process create communicate pipe set process term process group job signal send process group process usually inherit process group parent setpgrp system allow process change group process group c shell control operation mul- tiple

job process group use terminal device o time foreground job attention user terminal nonattached job background job perform function user interaction access terminal control process group signal job control terminal inherit parent process group control terminal match group process process foreground allow perform o. nonmatching background process attempt sigttin sigttou signal send process group signal usually cause process group freeze foreground user point receive sigcont signal indicate process perform o. similarly sigstop send foreground process group freeze system call exist set return interval timer getitimer()/ setitimer current time gettimeofday()/settimeofday microsecond addition process ask process identifier get- pid group identifier getgid machine execute gethostname value system interface unix support augment large collection library routine header file header file provide definition complex data structure system call addition large library function provide additional program support example unix o system call provide reading writing block byte application want read write 1 byte time possible require system byte high overhead instead set standard library routine standard o package access header file < stdio.h > provide interface read write thousand byte time local buffer transfer buffer user memory o desire formatted o support standard o package additional library support provide mathematical function net- work access datum conversion freebsd kernel support 300 system call c program library 300 library function library function eventually result system call necessary exam- ple getchar library routine result read system file buffer programmer generally need distin- guish

basic set kernel system call additional function provide library function programmer user unix system deal mainly set system program write available execution program necessary system call support function system call contain program need obvious user common system program group category file directory orient example system program manipulate directory mkdir create new directory rmdir remove directory cd change current directory pwd print absolute path current work directory ls program list name file current directory 28 option ask property file display example -l option ask long listing show file owner protection date time creation size cp program create new file copy exist file mv program move file place directory tree case simply require renaming file necessary file copy new location old copy delete file delete rm program make unlink display file terminal user run cat cat program take list file concatenate copy result standard output commonly terminal high speed cathode ray tube crt display course file speed fast read program display file screen time pause user type character continue screen head program display line file tail show line basic system program widely unix addition number editor ed sed emacs vi compiler c python fortran text formatter troff tex scribe program sort sort compare file cmp diff look pattern grep awk send mail user mail shell commands user write system program normally execute command interpreter command interpreter unix user process like note early call shell surround kernel operate system user write shell fact shell general use bourne shell write steve bourne probably widely widely available c shell work bill joy founder sun microsystems popular bsd system korn shell dave korn popular combine feature bourne shell c shell common shell share command language syntax unix normally interactive system shell indicate readiness accept command type prompt user type command single line instance line ls -l

prompt user type command single line instance line ls -l percent sign usual c shell prompt ls -l type user long list directory command command argument user type command line separate white space space tab command build shell cd typical command executable binary object file list directory search path keep shell command directory search path search order file file find load execute search path set user directory /bin /usr bin search path typical search path freebsd system /usr avi bin /usr local bin /bin /usr bin ls command object file /bin l shell /bin sh bourne shell /bin csh c shell execution command fork system follow execve object file shell usually wait suspend execution command complete figure c.4 simple syntax ampersand end command line indicate shell wait completion command command leave run manner shell continue interpret command say background command run background process shell wait say run foreground c shell freebsd system provide facility call job control par- tially implement kernel mention previously job control allow process move foreground background pro- cesse stop restart condition back- ground job want input user terminal scheme allow control process provide windowing layering interface require special hardware job control useful window system x window system develop mit window treat terminal allow multiple process foreground win- dow time course background process exist window korn shell support job control job control process group likely standard future version unix process open file like process expect file descriptor number 0 1 2 open start file descrip- tor inherit fork possibly execve create process know standard input 0

standard output 1 standard error 2 frequently open user terminal program read user type read standard input program send output user screen write standard output standard error file descriptor open writing error output standard output ordinary output program accept file terminal standard input standard output program care input come output go elegant design feature unix common shell simple syntax change file open standard o stream process change standard file call o redirection syntax o redirection show figure c.5 meaning command ls > filea direct output ls file filea pr < filea > fileb lpr < fileb program > err input filea output fileb input fileb save standard output standard error file standard /io/ redirection example ls command produce listing name file current directory pr command format list page suitable printer lpr command spool format output printer /dev lp0 subsequent command force output error message redirect file ampersand error message appear terminal pipeline filter shell script command figure c.5 coalesce ls | pr | lpr vertical bar tell shell arrange output precede command pass input follow command pipe carry datum process process write end pipe process read end example write end pipe set shell standard output ls read end pipe standard input pr pipe pr lpr acommand pr pass standard input standard output perform processing call filter unix command filter complicated function piece pipeline common command common function output formatting need build numerous command output program pipe pr appropriate filter common unix shell programming language shell variable usual high level programming language control con- struct loop conditional execution command analogous subroutine file shell command shell script execute like command appropriate shell invoke automatically read shell programming combine ordinary program conveniently sophisticated application need program- ming

conventional language external user view commonly think definition unix easily change definition write new shell different syntax semantic greatly change user view change kernel programmer interface menu drive iconic interface unix exist x window system rapidly standard heart unix course kernel kernel difficult change user interface program depend system call provide remain consistent course new system call add increase functionality program modify use new call major design problem operate system representation pro- cesse substantial difference unix system ease multiple process create manipulate process represent unix control block system con- trol block accessible virtual address space user process control block associate process store kernel kernel use information control block process control cpu scheduling process control block basic data structure associate process process structure process structure contain system need know process process swap unique process identifier scheduling information example priority process pointer control block array process structure length define system link time process structure ready process keep link scheduler doubly link list ready queue pointer process structure process parent young live child relative interest list process share program code text virtual address space user process divide text program code datum stack segment datum stack segment address space grow separately usually opposite direction frequently stack grow datum grow text segment intel 8086

separate instruction data space address space different datum stack usually read debugger put text segment read write mode allow insertion breakpoint process sharable text freebsd pointer process structure text structure text structure record process text segment include pointer list process structure page table text segment find disk swap text structure resident main memory array structure allocate system link time text datum stack segment process swap segment swap page page table record information mapping process virtual memory physical memory process structure contain pointer page table use process resident main memory address process swap device process swap special separate page table share text segment process share text segment entry page process page table information process need process resident swap keep user structure u structure process structure structure map read user virtual address space user process read content writable kernel user structure contain copy process control block pcb keep save process general register stack pointer program counter page table base register process run space system parameter return value user group identifier associate process effective user identifier keep process structure keep signal timer quota data structure obvious relevance ordinary user current directory table open file maintain process user system mode ordinary work user mode system perform system mode system user phase process execute simultaneously process execute system mode kernel stack process user stack belong process kernel stack process immediately follow user structure kernel stack user structure compose system datum segment process process instance interrupt handling figure c.6 illustrate process structure find part process fork system allocate new process structure new process identifier child process copy user structure ordinarily need new text structure process share text appropriate counter list merely update new page table construct new main memory allocate datum stack segment child process copying user structure preserve open file descriptor user group identifier signal

user structure preserve open file descriptor user group identifier signal handling similar property process vfork system copy datum stack new process new process simply share page table old new user structure new process structure create common use system occur shell execute command wait completion parent process use vfork produce child process child process wish use execve immediately change virtual address space completely need complete copy swappable process image system datum structure find part process process structure parent process data structure necessary manipulate pipe keep register vfork execve file close process affect process kernel datum structure involve depend user structure share parent suspend call vfork child call execve terminate parent change memory child need parent process large vfork produce substantial saving system cpu time fairly dangerous system memory change occur process execve occur alter- native share page duplicate page table mark entry page table copy write hardware protection bit set trap attempt write share page trap occur new frame allocate share page copy new frame page table adjust page long share long need write protect execution resume execve system create new process user structure text datum process replace open file preserve way specify certain file descriptor close execve signal handling property preserve arrangement specific user routine signal cancel obvious reason process identifier property process unchanged cpu scheduling unix design benefit interactive process process give small cpu time slice priority algorithm reduce round- robin scheduling cpu bind job process scheduling priority associate large number priority pzero kill signal ordinary user process positive priority likely run system process user process set precedence nice command cpu time process accumulate low positive priority vice versa negative feedback cpu scheduling make difficult single process cpu time process aging employ prevent starvation old unix system 1 second quantum round robin schedul- ing freebsd reschedule process 0.1 second

quantum round robin schedul- ing freebsd reschedule process 0.1 second recompute priority second round robin scheduling accomplish timeout mechanism tell clock interrupt driver kernel subroutine specified interval subroutine call case cause rescheduling resubmit timeout priority recomputation time subroutine resubmit timeout preemption process kernel process relinquish cpu wait o time slice expire process choose relinquish cpu go sleep event kernel primitive purpose call sleep confuse user level library routine sleep take argument convention address kernel datum structure relate event process wait event occur system process know call wakeup address correspond event process sleep address ready queue run example process wait disk o complete sleep address buffer header correspond datum transfer interrupt routine disk driver note transfer complete call wakeup buffer header interrupt use kernel stack process happen run time wakeup system process process actually run choose scheduler sleep take second argument scheduling priority purpose priority argument pzero prevent process awaken prematurely exceptional event signal signal generate leave pende system half affect process run event usually happen soon signal normally cause process awaken process wait condition memory associate event caller routine sleep event prepare deal premature return include possibility reason waiting vanish race condition involve event mechanism process decide check flag memory instance sleep event event occur process execute primitive sleep event process sleep sleep forever prevent situation raise hardware processor priority critical section interrupt occur process desire event run sleep hardware processor priority manner protect critical region kernel great obstacle port unix multiple processor machine problem prevent porting repeatedly process text editor o bind general schedule mainly basis wait o. experience suggest unix scheduler perform well o bind job observe cpu bind job text formatter language interpreter refer cpu scheduling correspond closely short term scheduling chapter 3 negative feedback property priority scheme provide long term scheduling largely determine long term job

scheme provide long term scheduling largely determine long term job mix medium term scheduling swapping mechanism describe section c.6 unix early development pdp-11 pdp-11 segment virtual address space size 8,192 byte large machine pdp-11/70 allow separate instruction address space effectively double address space number segment address space relatively small addition kernel severely constrain dedication data segment interrupt vector point process system datum segment unibus system o bus register small pdp-11s total physical memory limit 256 kb total memory resource insufficient justify support complex memory management algorithm unix swap entire process memory image berkeley introduce paging unix 3bsd vax 4.2bsd demand- page virtual memory system paging eliminate external fragmentation memory internal fragmentation occur negligible reason- ably small page size paging allow execution part process memory job keep main memory swapping keep minimum demand paging straightforward manner process need page page page fault kernel occur frame main memory allocate proper disk page read frame optimization page need page table process mark invalid page replacement process mark valid o transfer page similarly retrieve list free frame process start page prepage free list recovery mechanism arrangement process prepaging startup seldom result page- fault overhead close pure demand paging freebsd implement page coloring paging queue queue arrange accord size processor l1 l2 cache new page need allocate freebsd try optimally align cache page fetch disk lock memory duration transfer locking ensure page select page replacement page fetch map properly remain locked raw physical o page replacement algorithm interesting 4.2bsd use modifi- cation second chance clock algorithm describe section 10.4.5 map nonkernel main memory core map cmap sweep linearly repeatedly software clock hand clock hand reach give frame

frame mark use software condition example physical o progress frame free frame leave untouched clock hand sweep frame correspond text process page table entry frame locate entry invalid

correspond text process page table entry frame locate entry invalid frame add free list page table entry invalid reclaimable page time want valid bsd tahoe add support system implement reference bit system pass clock hand turn reference bit second pass place page reference bit remain free list replacement course page dirty write disk add free list pageout cluster improve check sure number valid data page process fall low page device flood request mechanism process limit main memory use lru clock hand scheme implement pagedaemon process 2 remember swapper process 0

init process 1 process spend time sleep check time second schedule timeout action necessary process 2 awaken number free frame fall threshold lotsfree pagedaemon awaken large free memory pagedaemon impose load system run sweep clock hand time pagedaemon process awak- ene number frame scan usually number page determine number frame lack reach lotsfree number frame scheduler deter-

mine need reason frame need long sweep number frame free rise lotsfree expect sweep complete hand stop pagedaemon process sleep parameter control range clock hand sweep determine system startup accord main memory pagedae- mon use 10 percent cpu time scheduler decide paging system overload process swap overload relieve swapping usually happen condition meet load average high free memory fall low limit minfree average memory available recent time desirable desfree lotsfree > desfree > minfree word chronic shortage memory process try run cause swapping free memory extremely low moment excessive paging rate need memory kernel enter calculation rare case process swap scheduler course reason simply run long parameter lotsfree usually quarter memory map clock hand sweep desfree minfree usually different system limit fraction available mem- ory freebsd dynamically adjust paging queue virtual memory parameter rarely need adjust minfree page keep free order supply page need interrupt time process text segment default share read scheme practical paging external fragmentation swap space gain share offset negligible overhead involve kernel virtual space large cpu scheduling memory swapping paging interact low priority process likely page page likely swap entirety age preference choose process swap guard thrashing paging effectively ideally process swap idle process need small working set page main memory time pagedaemon reclaim unused page use process memory process need fraction process total virtual size half process swap long time unix file system support main object file directory directo- rie file special format representation file basic block fragment file system take datum block contain user file let consider data block store hardware disk sector usually 512 byte block size large 512 byte desirable speed unix file system usually contain large number small file large block cause excessive internal fragmentation early 4.1bsd file system limit 1,024 byte 1 kb block 4.2bsd solution use block size file

byte 1 kb block 4.2bsd solution use block size file indirect block block file large size 8 kb block appropriate multiple small fragment size example 1,024 kb fill file file size 18,000 byte 8 kb block 2 kb fragment fill completely block fragment size set file system creation accord intend use file system small file expect fragment size small repeat transfer large file expect basic block size large implementation detail force maximum block fragment ratio 8:1 minimum block size 4 kb typical choice 4,096:512 case 8,192:1,024 suppose datum write file transfer size 1 kb byte block fragment size file system 4 kb 512 byte file system allocate 1 kb fragment contain datum transfer transfer cause new 2 kb fragment allocate datum original fragment copy new fragment follow second 1 kb transfer allocation routine attempt find require space disk immediately follow exist fragment copying necessary seven copy require fragment block provision program discover block size file transfer size avoid fragment recopying file represent inode record store infor- mation specific file disk figure c.7 inode pro- nounce eye node derive index node originally spell node hyphen fall use year term spell node size block count unix inode inode contain user group identifier file time file modification access count number hard link directory entry file type file plain file directory symbolic link character device block device socket addition inode contain 15 pointer disk block contain datum content file 12 pointer point direct block contain address block contain datum file datum small file 12 block reference immediately copy inode keep main memory file open block size 4 kb 48 kb datum access directly inode pointer inode point indirect block file large use indirect block indirect block major block size fragment size apply datum block indirect block pointer address single indirect block single indirect block

indirect block pointer address single indirect block single indirect block index block contain datum address block contain datum

double indirect block pointer address block contain address block contain pointer actual datum block pointer contain address triple indirect block need minimum block size file system 4.2bsd 4 kb file 232 byte use double triple indirection block pointer take 4 byte 49,152 byte accessible direct block 4,194,304 byte accessible single indirection 4,294,967,296 byte reachable double indirection total 4,299,210,752 byte large 232 byte number 232 significant file offset file structure main memory keep 32 bit word file large 232 byte file pointer sign integer seek backward forward file actual maximum file size 2321 byte gigabyte large purpose plain file distinguish directory level implementa- tion directory content keep data block directory represent inode way plain file inode type field distin- guishe plain file directory plain file assume structure directory specific structure version 7 file name limit 14 character directory list 16 byte entry 2 byte inode number 14 byte file freebsd file name variable length 255 byte directory entry variable length entry contain length entry file inode number variable length entry make directory management search routine complex allow user choose meaningful name file direc- tory name directory new directory entry add directory space available generally exist file linear search user refer file path file system use inode definition file kernel map supply user path inode directory mapping start directory determine mention early character path start directory root directory path start character slash start directory current directory current process start directory check proper file type access permission error return necessary inode start directory available element path end path file start directory search error return find path element current inode refer directory error return access deny directory search way previous process continue end path reach desire inode return step step process need directory mount point symbolic link discuss encounter cause translation different directory structure

point symbolic link discuss encounter cause translation different directory structure hard link simply directory entry like handle symbolic link start search path take content symbolic link prevent infinite loop count number symbolic link encounter path search return error limit exceed nondisk file device data block allocate disk kernel notice file type indicate inode call appropriate driver handle o inode find instance open system file structure allocate point inode file descriptor give user refer file structure freebsd directory cache hold read 4 table open file system control block recent directory inode translation greatly increase file system per- map file descriptor inode system refer open file indicate file pass file descriptor argument file descriptor kernel index table open file current process entry table contain pointer file structure file structure turn point inode figure c.8 open file table fix length settable boot time fix limit number concurrently open file system read write system call position file argument kernel keep file offset update appropriate read write accord number datum actually transfer offset set directly lseek system file descriptor index array inode pointer instead file pointer offset keep inode process open file process need offset file keep offset inode inappropriate file structure contain offset file structure inherit child process fork process share offset location inode structure point file structure core copy inode disk core inode extra field reference count file structure point file structure similar reference count file descriptor refer count zero entry long need reclaim reuse file system user see support datum mass storage device usually disk user ordinarily know file system logical file system actually consist physical file system different device device characteristic differ separate hardware device define physical file system fact generally want partition large physical device disk multiple logical device logical device define physical file system figure c.9 illustrate directory structure

device define physical file system figure c.9 illustrate directory structure partition file system map logical device partition physical device size location partition code device driver early system maintain disk freebsd partition physical device multiple file system bene- fit different file system support different use partition file system need swap area virtual memory software reliability improve software dam- age generally limit file system improve efficiency vary file system parameter block fragment size partition have separate file system prevent program available space large file file split file system finally disk backup partition fast search backup tape file partition small restore partition tape fast number file system drive vary accord size disk purpose computer system file system logical file system mapping logical file system physical device root file system available file system mount integrate directory hierarchy root file system bit inode structure indicate inode file system mount reference file cause mount table search find device number mount device device number find inode root directory mount file system inode conversely path element directory search root directory file system mount mount table search find inode mount inode file system separate system resource represent set file sector logical device boot block possibly contain primary bootstrap program secondary bootstrap program reside 7.5 kb system need partition con- taine boot block datum system manager install duplicate privileged program allow booting primary copy damage superblock contain static parameter file system parameter include total size file system block fragment size datum block assorted parameter affect allocation policy user interface file system simple define allow implementation file system change significant effect user file system change version 6 version 7 3bsd version 7 4bsd version 7 size inode double maximum file file system size increase detail free list handling superblock information change time seek 16 bit offset lseek 32 bit offset allow

time seek 16 bit offset lseek 32 bit offset allow specification offset large file change visible outside 4.0bsd size block file system increase 512 byte 1,024 byte increase size produce increase internal fragmentation disk double throughput mainly great number datum access disk transfer idea later adopt system v number idea device driver program 4.2bsd add berkeley fast file system increase speed accompany new feature symbolic link require new system call long file name necessitate new directory system call traverse now- complex internal directory structure finally truncate call add fast file system success find implementation unix performance possible layout allocation policy discuss section 14.4.4 discuss change sunos increase disk throughput layout allocation policy kernel use < logical device number inode number > pair identify file logical device number define file system involve inode file system number sequence version 7 file system inode array immediately follow single superblock beginning logical device data block follow inode inode number effectively index array version 7 file system block file disk end inode array end file system free block keep link list superblock block push free list remove need serve new file extend existing file block file arbitrarily far inode furthermore file system kind disorganized block file reverse process reinitialize restore entire file system convenient task perform process describe section 14.7.4 difficulty reliability file system suspect speed superblock mount file system keep memory keep superblock memory allow kernel access superblock quickly especially free list 30 second superblock write disk core disk copy synchronize update program sync system system bug hardware failure cause system crash destroy core superblock update disk free list disk accurately reflect state disk reconstruct perform lengthy examination block file system problem remain new 4.2bsd file system implementation radically different version 7 reimplementation primarily improve efficiency robustness change invisible outside kernel change introduce time improve efficiency robustness change invisible outside kernel change introduce time visible system user level example include symbolic link long file name 255 character change require feature kernel program use space allocation especially different major new concept freebsd cylinder group cylinder group introduce allow localization block file cylinder group occupy consecutive cylin- der disk disk access cylinder group require minimal disk head movement cylinder group superblock cylinder block array inode datum block figure c.10 4.3 bsd cylinder group superblock cylinder group identical superblock recover event disk corruption cylinder block contain dynamic parameter particular cylinder group include bit map free datum block fragment bitmap free inode statistic recent progress allocation strategy keep header information cylinder group superblock cylinder block inode beginning group header information cylinder group disk platter single disk head crash wipe cylinder group header information different offset beginning group directory listing command ls commonly read inode file directory make desirable inode close disk reason inode file usually allocate cylinder group contain inode file parent directory localize inode new directory different cylinder group parent directory cylinder group choose new directory inode great number reduce disk head seek involve access data block file allocate block cylinder group possible single file allow block cylinder group file exceed certain size 2 mb block allocation redirect different cylinder group new group choose have average free space file continue grow allocation redirect megabyte cylinder group block small file likely cylinder group number long head seek involve access large file keep small level disk block allocation routine global policy routine select desire disk block accord consideration discuss local policy routine use specific information record cylinder block choose block near request request block use return routine return block rotationally close request cylinder block different cylinder cylinder group block cylinder group quadratic rehash cylinder group find block fail exhaustive

cylinder group quadratic rehash cylinder group find block fail exhaustive search free space typically 10 percent leave file system block usually find desire quadratic rehash exhaustive search performance file system degrade use increase efficiency fast file system typical disk utilize 30 percent raw transfer capacity percentage marked improvement realize version 7 file system 3 percent bandwidth bsd tahoe introduce fat fast file system allow number inode cylinder group number cylinder cylinder group number distinguish rotational position set file system create freebsd previously set parameter accord disk hardware type purpose operate system hide peculiarity specific hardware device user example file system present simple consistent storage facility file independent underlie disk hardware unix peculiarity o device hide bulk kernel o system o system consist buffer cache system general device driver code driver specific hardware device device driver know peculiarity specific device major part o system diagram figure c.11 main kind o freebsd block device character device socket interface socket interface pro- tocol network interface describe section c.9.1 block device include disk tape distinguish characteristic directly addressable fix block size usually 512 byte block device driver require isolate detail track cylinder rest kernel block device accessible directly appropriate device special file /dev rp0 commonly access indirectly file system case transfer buffer block buffer cache profound effect efficiency character device include terminal line printer include network interface use block buffer cache instance /dev mem interface physical main memory /dev null bottomless sink datum endless source end file marker device high speed graphic interface buffer o directly user data space use block buffer cache class character device terminal terminal like device use c list buffer small block buffer cache block device character device main device class device driver access array entry point array block device character device device distinguish class block character device number device

character device device distinguish class block character device number device number consist part major device number index array character block device find entry appropriate device driver minor system interface kernel 4.3 bsd kernel o structure device number interpret device driver example logical disk partition terminal line adevice driver connect rest kernel entry point record array class use common buffering system segregation important portability system configuration block buffer cache block device mention use block buffer cache buffer cache consist number buffer header point piece physical memory device number block number device buffer header block currently use keep link list follow buffer recently link lru order lru list buffer recently valid content age list buffer physical memory associate buffer list hash device block number search block want device read cache search block find o transfer necessary find buffer choose age list list lru list device number block number associate update memory find necessary new datum transfer device buffer lru buffer write device modify buffer reuse write block question buffer cache new datum buffer overwrite previous datum buffer header mark indicate buffer modify o immediately necessary datum write buffer need datum block find buffer cache buffer choose read transfer buffer write periodically force dirty buffer block minimize potential file system inconsistency number datum buffer freebsd variable maximum file system usually 8 kb minimum size paging cluster size usually 1,024 byte buffer page cluster align page cluster map buffer time disk block map buffer time list hold buffer header physical memory block 8 kb split hold multiple small block header need block retrieve number datum buffer grow user process write datum follow buffer increase occur new buffer large hold datum allocate original datum copy follow new datum buffer shrink buffer take queue excess page buffer release write disk device magnetic tape require block write

buffer release write disk device magnetic tape require block write certain order facility provide force synchronous write buffer device correct order directory block write synchronously forestall crash inconsistency consider chaos occur change directory directory entry update size buffer cache profound effect performance system large percentage cache hit high number actual o transfer low freebsd optimize buffer cache continually adjust memory program disk cache interesting interaction occur buffer cache file system disk driver datum write disk file buffer cache disk driver sort output queue accord disk address action allow disk driver minimize disk head seek write datum time optimize disk rotation synchronous write require process write disk simply write buffer cache system asynchronously write datum disk convenient user process see fast write datum read disk file block o system read ahead write near asynchronous read output disk file system fast input large transfer counter intuition raw device interface block device character interface call raw device interface interface differ block interface block buffer cache bypass disk driver maintain queue pende transfer record queue specify read write give main memory address transfer usually 512 byte increment device address transfer usually address disk sector transfer size sector simple map information block buffer require simple map piece main memory correspond user process virtual address space mapping raw disk interface instance unbuffered transfer directly user virtual address space allow size transfer limit physical device require number byte kernel accomplish transfer swapping paging simply put appropriate request queue appropriate device special swapping paging device driver need 4.2bsd file system implementation actually write largely test user process raw disk interface code move kernel interesting face mach operate system file system se file system implement user level task appendix d mention terminal terminal like device use character buffer system keep small block character usually 28 byte link list call

keep small block character usually 28 byte link list call c list free character buffer keep single free list device driver use limit number character queue time give terminal line routine enqueue dequeue character list write system terminal enqueue character list device initial transfer start interrupt cause dequeuing character input similarly interrupt drive terminal driver typically support input queue conversion raw queue canonical queue trigger interrupt routine put end line awaken system phase conversion character canonical queue available return user process device driver bypass canonical queue return character directly raw queue mode operation know raw mode screen editor program need react keystroke use mode task accomplish isolate process require interprocess communication isolated computing system long serve application networking increasingly important fur- thermore increase use personal workstation resource sharing common interprocess communication traditionally unix strong point pipe discuss section c.4.3 ipc mechanism characteris- tic unix pipe permit reliable unidirectional byte stream process traditionally implement ordinary file excep- tion file system create instead pipe system size fix process attempt write pipe process suspend datum previously write pipe read writing continue beginning file pipe true circular buffer benefit small size pipe usually 4,096 byte pipe datum seldom actually write disk usually keep memory normal block buffer cache freebsd pipe implement special case socket mecha- nism socket mechanism provide general interface facility pipe local machine network facility machine pipe process relate use fork system socket mechanism socket endpoint communication socket use usually address bind nature address depend communication domain socket characteristic property domain process communicate domain use address format single socket communicate domain domain currently implement freebsd unix domain af unix internet domain af inet xerox network services ns domain af ns address format unix domain ordinary file system path /alpha beta gamma process commu- nicate internet domain use darpa internet communication protocol tcp

commu- nicate internet domain use darpa internet communication protocol tcp ip internet address consist 32 bit host number 32 bit port number represent rendezvous point host socket type represent class service type implement communication domain type implement give domain implement protocol select user stream socket socket provide reliable duplex sequence datum stream datum lose duplicate delivery record boundary type support internet domain tcp unix domain pipe implement pair communicate stream sequence packet socket socket provide data stream like stream socket record boundary provide type xerox af ns protocol datagram socket socket transfer message variable size direction guarantee message arrive order send unduplicate arrive original message record size preserve datagram arrive type support internet domain udp reliably deliver message socket socket transfer message guarantee arrive like message trans- ferre datagram socket type currently unsupported raw socket socket allow direct access process proto- col support socket type protocol accessible include uppermost one low level protocol example internet domain possible reach tcp ip beneath ethernet protocol beneath capability useful develop set system call specific socket facility socket system create socket take argument specification communication domain socket type protocol support type value return small integer call socket descriptor occupy space file descriptor socket descriptor index array open file u structure kernel file structure allocate freebsd file structure point socket structure instead inode case certain socket information socket type message count datum input output queue keep directly socket structure process address socket socket bind socket bind system take socket descriptor pointer length byte string content length byte string depend address format connect system initiate connection argument syntactically bind socket descriptor represent local socket address foreign socket attempt connect process communicate socket ipc follow client server model model server process provide service client process service available server process listen well- know address client process use connect

server process listen well- know address client process use connect reach server server process use socket create socket bind bind know address service socket use lis- system tell kernel ready accept connection client specify pending connection kernel queue server service finally server use accept sys- tem accept individual connection listen accept argument socket descriptor original socket system accept return new socket descriptor correspond new connec- tion original socket descriptor open connection server usually use fork produce new process accept service client original server process continue listen connection system call set parameter connection return address foreign socket accept connection socket type stream socket establish address endpoint know address infor- mation need transfer datum ordinary read write system call transfer datum simple way terminate connection destroy associate socket use close system socket descriptor wish terminate direction communication duplex connection shutdown system purpose socket type datagram socket support connection instead socket exchange datagram address individually system call sendto recvfrom connection argument socket descriptor buffer pointer length address buffer pointer length address buffer contain appropriate address sendto fill address datagram receive recvfrom number datum actually transfer return system call select system multiplex datum transfer several file descriptor and/or socket descriptor allow server process listen client connection service fork process connection connection server socket bind listen service select socket descriptor select indicate activity descriptor server accept fork process new descriptor return accept leave parent process select current unix system support uucp network facility dial telephone line support uucp mail network usenet news network rudimentary networking facility support

remote login remote proce- dure call distribute file system facility completely implement user process operate system freebsd support darpa internet protocol udp tcp ip icmp wide range ethernet token ring arpanet interface framework kernel support protocol intend facilitate implementa- tion protocol protocol accessible socket interface rob gurwitz bbn write version code add package international standards organization iso open system intercon- nection osi reference model networking prescribe seven layer net- work protocol strict method communication imple- mentation protocol communicate peer entity speak protocol layer protocol protocol interface protocol layer immediately system iso networking model implement freebsd reno 4.4bsd freebsd networking implementation certain extent socket facility oriented arpanet reference model arm arpanet original form serve proof concept network- e idea packet switching protocol layering arpanet retire 1988 hardware support long state art successor nsfnet internet large serve communication utility researcher test bed internet gateway research arm predate iso model iso model large inspire arpanet research iso model interpret set limit pro- tocol communicate layer arm allow protocol layer protocol layer arm process application layer subsume application presentation session layer iso model user level program file- transfer protocol ftp telnet remote login exist level host host layer correspond iso transport network layer transmission control protocol tcp internet protocol ip layer tcp ip tcp correspond iso transport protocol ip perform addressing function iso network layer network interface layer span low iso network layer entire data link layer protocol involve depend physical network type arpanet use imp host protocol ethernet use ethernet protocol network hardware arm primarily concern software explicit network hardware layer actual network hardware correspond iso physical layer networking framework freebsd generalized iso model arm closely relate arm figure user process communicate network protocol process machine socket facility facility correspond iso session layer responsible set control socket support protocol possibly layer aprotocol provide

responsible set control socket support protocol possibly layer aprotocol provide service reliable delivery suppression duplicate transmission flow control addressing depend socket type support service require high protocol protocol communicate protocol network interface appropriate network hardware little restriction general framework protocol communicate protocol protocol layer user process mean raw socket type directly access layer protocol uppermost support socket type stream raw network interface capability route process new protocol development network interface driver network controller type network interface responsible handle characteristic specific local network address arrangement ensure protocol interface need concern characteristic function network interface depend largely network hardware necessary network network network reference model layering support reliable transmission level network provide broadcast addressing socket facility networking framework use common set memory buffer mbuf intermediate size large buffer block o system c list character device mbuf 128 byte long 112 byte datum rest pointer link mbuf queue indicator datum area actually use datum ordinarily pass layer socket protocol protocol protocol protocol network interface mbuf ability pass buffer contain datum eliminate datum copying frequently need remove add protocol header convenient efficient purpose able hold datum occupy area size memory management page datum mbuf reside mbuf memory mbuf page table purpose pool page dedicate mbuf use early advantage unix write high level language distribute source form provide powerful operating- system primitive inexpensive platform advantage lead unix popularity educational research government institution eventu- ally commercial world popularity produce strain unix vary improve facility unix provide file system tree structured directory kernel support file unstructured sequence byte direct access sequential access support system call library routine file store array fix size

datum block trail fragment data block find pointer inode directory entry point inode disk space allocate cylinder group minimize head movement improve performance unix multiprogrammed system process easily create new pro- cesse fork system process communicate pipe generally socket group job control process represent structure process structure user structure cpu scheduling priority algorithm dynamically compute priority reduce round robin scheduling extreme freebsd memory management use swapping support paging pagedaemon process use modify second chance page replacement algo- rithm free frame support execute process page file o use block buffer cache minimize actual o. terminal device use separate character buffer system networking support important feature freebsd socket concept provide programming mechanism access process network socket provide interface set mckusick et al 2015 provide good general discussion freebsd modern scheduler freebsd describe roberson 2003 lock multithreaded freebsd kernel describe baldwin 2002 freebsd describe freebsd handbook download j. baldwin locking multithreaded freebsd kernel usenix bsd 2002 mckusick et al 2015 m. k. mckusick g. v. neville neil r. n. m. wat- son design implementation freebsd unix operating system second edition pearson 2015

j. roberson ule modern scheduler freebsd pro- ceeding usenix bsdcon conference 2003 page 17–28 chapter firs write 1991 update time long modify appendix examine mach operating system mach design incorporate recent innovation operating system research produce fully functional technically advanced system unlike unix develop regard multiprocessing mach incorporate mul- tiprocessing support support exceedingly flexible accom- modate share memory system system memory share processor mach design run computer system range processor thousand processor addition easily port varied computer architecture key goal mach distribute system capable function heterogeneous hardware experimental operating system design build mach satisfy need user well offer compatibility unix 4.3 bsd compatibility give unique opportunity compare functionally similar internally dissimilar operate system mach unix differ emphasis mach discussion exactly parallel unix discussion addition include section user interface component similar user interface 4.3 bsd mach provide ability layer emulation operate system operating system run concurrently mach history mach system mach trace ancestry accent operate system develop carnegie mellon university cmu accent pioneer number novel operating system concept utility limit inability execute unix application strong tie single hardware architecture difficult port mach communication system philosophy derive accent significant portion system example virtual memory system management task thread develop scratch important goal mach effort support multiprocessor mach system mach development follow evolutionary path bsd unix sys- tem mach code initially develop inside 4.2bsd kernel bsd ker- nel component replace mach component mach component complete bsd component update 4.3 bsd available 1986 virtual memory communication subsystem run dec vax computer family include multiprocessor version shortly 1987 see completion encore multimax sequent balance multiprocessor version include task thread support official release system release 0 release 1 release 2 mach provide compatibility corresponding bsd system include bsd code kernel new feature capability mach kernel release large correspond bsd kernel mach 3 figure d.1 move bsd

large correspond bsd kernel mach 3 figure d.1 move bsd code outside kernel leave small microkernel system implement basic mach feature kernel unix specific code evict run user mode server exclude unix specific code kernel allow replacement bsd operating system simultaneous execution multiple operating system interface microkernel addition bsd user mode implementation develop dos macintosh operate system osf/1 approach similarity virtual machine concept virtual machine define software mach kernel interface hardware release 3.0 mach available wide variety system include single processor sun mach propel forefront industry attention open software foundation osf announce 1989 use mach 2.5 basis new operate system osf/1 release osf/1 occur year later compete unix system v release 4 operate system choice unix international ui member osf member include key operate system workstation brainchild steve jobs apple computer fame osf evaluate mach 3 basis future mach 3 structure operating system release research mach continue cmu osf mach operate system design provide basic mechanism current operate system lack goal design operate system bsd compatible addition excel following area support diverse architecture include multiprocessor vary degree share memory access uniform memory access uma non- uniform memory access numa remote memory access norma ability function vary intercomputer network speed wide area network high speed local area network tightly couple simplified kernel structure small number abstraction turn abstraction sufficiently general allow operating system implement mach distributed operation provide network transparency client object orient organization internally externally integrated memory management interprocess communication provide efficient communication large number datum communication base memory management heterogeneous system support mach widely available inter- operable computer system multiple vendor designer mach heavily influence bsd unix general benefit include simple programmer interface good set primitive con- sistent set interface system facility easy portability wide class single processor extensive library utility application ability combine utility easily pipe course designer

library utility application ability combine utility easily pipe course designer want redress see drawback bsd akernel repository redundant feature consequently difficult manage modify original design goal difficult provide support multiprocessor distribute system share program library instance kernel design single processor provision lock code datum processor mach system fundamental abstraction provide similar compet- e mean accomplish task development mach continue huge undertaking benefit system equally large operate system run exist single processor multiprocessor architecture easily port future one make research easy computer scientist add feature user level code instead have write tailor operate system area experimentation include operat- e system database reliable distribute system multiprocessor language security distribute artificial intelligence current version mach system usually efficient major version unix perform achieve design goal mach developer reduce operating- system functionality small set basic abstraction functionality derive mach approach place little possible kernel powerful feature implement user level mach design philosophy simple extensible kernel concen- trate communication facility instance request kernel data movement process handle communication mechanism mach able provide system wide protection user protect communication mechanism optimize com- munication path result significant performance gain simple try optimize path mach extensible tradi- tionally kernel base function implement user level server instance pager include default pager implement exter- nally call kernel user mach example object orient system datum operation manipulate datum encapsulate abstract object operation object able act entity define detail operation implement hide internal datum structure programmer use object invoke define export operation programmer change internal opera- tion change interface definition change optimization affect aspect system operation object orient approach support mach allow object reside network mach system transparent user port mechanism discuss later section make possible mach primitive abstraction heart system follow task execution environment provide basic unit resource allocation consist virtual address space protect access system resource port contain thread thread

address space protect access system resource port contain thread thread basic unit execution run context task provide address space thread task share task resource port memory notion process mach traditional process implement task single thread control port basic object reference mechanism mach imple- mente kernel protect communication channel communication accomplish send message port message queue destination port thread immediately ready receive port protect kernel manage capability port right task port right send message port programmer invoke operation object send message port associate object object represent port receive message port set group port share common message queue thread receive message port set service multiple port receive message identify individual port set receive receiver use identify object refer message message basic method communication thread mach type collection datum object object contain actual datum pointer line datum port right pass message way task pass port right share memory work mach kernel permit new task use right obtain manner memory object source memory task access map portion object entire object address space object manage user mode external memory manager example file manage file server memory object object memory map access make sense map buffer implementation unix pipe example figure d.2 illustrate abstraction explain remainder chapter unusual feature mach key system efficiency blending memory interprocess communication ipc feature distribute system solaris nfs feature special purpose extension file system extend network mach provide general purpose extensible merger memory message heart kernel feature allow mach distributed parallel programming help implementation kernel mach connect memory management ipc allow implementation memory management base use memory object memory object represent port port ipc message send port request operation example pagein pageout object ipc memory object reside remote system access transparently kernel cache content memory object local memory conversely memory management tech- nique implementation message passing possible mach system mach basic

tech- nique implementation message passing possible mach system mach basic abstraction mach pass message move pointer share memory object copy object ipc tend involve considerable system overhead intrasystem mes- sage generally efficient communication accomplish share memory mach message base kernel message handling carry efficiently inefficiency message handling traditional operate system copying message task intracomputer message low network transfer speed intercomputer message solve problem mach use virtual memory remapping transfer content large message word message transfer modify receive task address space include copy message content virtual copy copy write technique avoid delay actual copying datum approach increase flexibility memory management user program great generality allow virtual copy approach tightly loosely couple computer improved performance unix message passing easy task migration port location independent task port move machine task previously communicate move task continue reference task port communicate message port section follow detail operation process management ipc memory management discuss mach chameleonlike ability support multiple operate system interface task think traditional process instruc- tion pointer register set task contain virtual address space set port right accounting information task passive entity thread execute task contain thread similar unix process fork system produce new unix process mach create new task fork new task memory duplicate parent address space dictate inheritance attribute parent memory new task contain thread start point create fork parent thread task suspend resume thread especially useful server application common unix task multiple thread service multiple request task thread allow efficient use

parallel compute resource have process processor correspond per- formance penalty operating system overhead task thread spread parallel processor thread add efficiency user level pro- gram instance unix entire process wait page fault occur system execute task multiple thread thread cause page fault execute system delay thread continue execute mach kernel- support thread chapter 4 thread cost associate support data structure kernel com- plex kernel scheduling algorithm provide algorithm thread state discuss chapter 4 user level thread state run thread execute wait allocate pro- cessor thread consider run block kernel wait page fault satisfy instance suspend thread execute processor wait allocate processor thread resume execution return run state state associate task operation task affect thread task suspend task involve suspend thread task thread suspension separate independent mechanism resume thread suspend task resume mach provide primitive thread synchronization tool build practice consistent mach philosophy provide mini- mach system mum sufficient functionality kernel mach ipc facility synchronization process exchange message rendezvous point thread level synchronization provide call start stop thread appropriate time suspend count keep thread count allow multiple suspend call execute thread equal number resume call occur thread resume unfor- tunately feature limitation error start execute stop suspend count negative routine synchronize shared datum access wait signal operation associate semaphore synchronization implement ipc call discuss method section d.5 c threads package mach provide low level flexible routine instead polished large restrictive function make programmer work low level mach provide high level interface program c language instance c thread package provide multiple thread control share variable mutual exclusion critical section condition variable synchronization fact c thread major influence posix pthreads standard operating system support result strong similarity c thread pthreads programming interface thread control routine include call perform task create new thread task give function execute param- eter input thread execute concurrently

task give function execute param- eter input thread execute concurrently create thread receive thread identifier return destroy call thread return value create thread wait specific thread terminate allow call thread continue synchronization tool like unix wait yield use processor signal scheduler run thread point useful presence preemptive scheduler relinquish cpu voluntarily time quantum scheduling interval expire thread use mutual exclusion achieve use spinlock discuss chapter 6 routine associate mutual exclusion routine mutex alloc dynamically create mutex variable routine mutex free deallocate dynamically create mutex vari- routine mutex lock lock mutex variable execute thread loop spinlock lock attain deadlock result thread lock try lock mutex variable bounded waiting guarantee c thread package dependent hardware instruction implement mutex routine routine mutex unlock unlock mutex variable like typical signal operation semaphore general synchronization busy waiting achieve use condition variable implement monitor describe chapter 6 condition variable associate mutex variable reflect boolean state variable routine associate general synchronization routine condition alloc dynamically allocate condition vari- routine condition free delete dynamically create condition variable allocate result condition alloc routine condition wait unlock associate mutex variable block thread condition signal execute condition variable indicate event wait occur mutex variable lock thread continue condition signal guarantee condition hold unblocked thread finally return condition wait awaken thread loop execute condition wait routine unblock condition hold example c thread routine consider bound buffer synchronization problem section 7.1.1 producer consumer represent thread access common bound buffer pool use mutex variable protect buffer update exclusive access buffer use condition variable block producer thread buffer block consumer thread buffer chapter 6 assume buffer consist n slot capable hold item mutex semaphore provide mutual exclusion access buffer pool initialize value 1 semaphore count number buffer respectively semaphore initialize value n semaphore initialize value 0 condition variable nonempty true buffer item nonfull true buffer

0 condition variable nonempty true buffer item nonfull true buffer slot step include allocation mutex condition variable condition alloc(nonempty nonfull code producer thread show figure d.3 code consumer thread show figure d.4 program terminate mutex condition variable need deallocate condition free(nonempty nonfull mach system produce item nextp condition wait(nonfull mutex add nextp buffer structure producer process cpu scheduler cpu scheduler thread base multiprocessor operate system complex process base relative generally thread multithreaded system process multitaske system keep track multiple processor difficult relatively new area research mach use simple policy scheduler manageable thread schedule knowledge task need scheduler thread compete equally resource include time quantum thread associate priority number range 0 127 base exponential average usage cpu thread recently cpu large time low condition wait(nonempty mutex remove item buffe nextc consume

item nextc structure consumer process priority mach use priority place thread 32 global run queue queue search priority order wait thread processor idle mach keep processor local run queue alocal run queue thread bind individual processor instance device driver device connect individual cpu run cpu instead central dispatcher assign thread processor processor consult local global run queue select appropriate thread run thread local run queue absolute priority global queue assume perform chore kernel run queue like object mach lock modify avoid simultaneous change multiple processor speed dispatching thread global run queue mach maintain list idle processor additional scheduling difficulty arise multiprocessor nature mach fix time quantum appropriate instance few runnable thread available processor wasteful interrupt thread context switch kernel thread quantum run thread place right running state instead fix length quantum mach vary size time quantum inversely total number thread system keep time quantum entire system constant example system 10 processor 11 thread 100 millisecond quantum context switch need occur processor second maintain desire quantum course complication exist relinquish cpu

processor second maintain desire quantum course complication exist relinquish cpu wait- e resource difficult traditional operate system thread issue alert scheduler thread block alert avoid race condition deadlock occur execution take place multiprocessor environment second actually cause thread move run queue appropriate event occur scheduler use internal thread state control mach design provide single simple consistent exception handle system support standard user define exception avoid redundancy kernel mach use kernel primitive possible instance exception handler thread task exception occur remote procedure rpc message synchro- nize execution thread cause exception victim handler communicate information exception victim handler mach exception emulate bsd signal disruption normal program execution come variety internally generate exception external interrupt interrupt asynchronously generate disruption thread task exception cause occurrence unusual condition thread execution mach general- purpose exception facility error detection debugger support mach system facility useful function take core dump bad task allow task handle error arithmetic emulate instruction implement hardware mach support different granularity exception handling error handling support thread exception handling debugger use task handling make little sense try debug thread exception multiple thread invoke multiple debugger aside distinction difference type exception lie inheritance parent task task wide exception handling facility pass parent child task debugger able manipulate entire tree task error handler inherit default handler thread- task creation time finally error handler precedence debugger exception occur simultaneously reason approach error handler normally task execute normally presence debugger exception handling proceed follow victim thread cause notification exception occurrence raise rpc message send handler victim call routine wait exception handle handler receive notification exception usually include infor- mation exception thread task cause exception handler perform function accord type exception handler action involve clear exception cause victim resume terminate victim thread support execution bsd program mach need support bsd- style signal signal provide software generate interrupt exception unfortunately

bsd- style signal signal provide software generate interrupt exception unfortunately signal limited functionality multithreaded operating system problem unix signal handler routine process receive signal signal cause problem process example division 0 problem remedie process limit access context second trou- blesome aspect signal design single threaded program instance make sense thread task signal signal see thread signal system work correctly multithreaded appli- cation mach run 4.3 bsd program signal abandon produce functionally correct signal package require rewrite code final problem unix signal lose loss occur signal type occur handle mach exception queue result rpc implementation externally generate signal include send bsd process process bsd server section mach 2.5 kernel behavior bsd hardware exception different matter bsd program expect receive hardware exception signal hardware exception cause thread arrive thread signal result produce hardware exception convert exception rpc task thread explicit use mach exception handling facility destination rpc default kernel task task purpose thread run continuous loop receive exception rpc rpc convert exception appropriate signal send thread cause hardware exception complete rpc clear original exception condition completion rpc initiate thread reenter run state immediately see signal execute signal handling code manner hardware exception begin uniform way exception rpc thread design handle exception receive exception standard bsd system signal mach 3.0 signal handling code move entirely server overall structure flow control similar mach 2.5 commercial operating system unix provide communication process host fix global name internet address location independence facility remote system need use facility know system provide facility usually datum message untyped stream byte mach simplify picture send message location independent port message contain type datum ease interpretation bsd communication method implement simplified system component mach ipc port message every- thing mach object object address commu- nication port message send port initiate operation object routine implement object depend port message communication mach deliver

object routine implement object depend port message communication mach deliver location independence object security communication data independence provide netmsgserver task section d.5.3

mach ensure security require message sender receiver right aright consist port capability send receive port like capability object orient system task receive right give port task send right object create creator allocate port represent object obtain access right port right give creator object include kernel pass message holder receive right send right message receiver message gain right sender lose task allocate port allow access object own communication destruction port holder receive right cause revocation right port task hold send right notify desire port implement protected bounded queue kernel system object reside queue sender abort mach system send wait slot available queue kernel deliver system call provide port following functionality allocate new port specify task caller task access right new port port return deallocate task access right port task hold receive right port destroy task send right potentially current status task port create backup port give receive right port task contain receive right request deallocation terminate task create kernel create port function task self return port represent task call kernel instance allocate new port task call port allocate task self task port thread creation result similar thread self thread kernel port scheme similar standard process id concept find unix port return task notify port kernel send event notification message notification port termination port collect port set facility useful thread service request come multiple port example multiple object port member port set time furthermore port set directly receive message instead message route port set queue port set pass message unlike port port set object serve purpose similar 4.3 bsd select system efficient message consist fix length header variable number type datum object header contain destination port reply port return message send length message figure d.5

datum message line datum limit 8 kb mach 2.5 system mach 3.0 limit data section simple type number character port right pointer line datum section associate type receiver unpack datum correctly use byte order different sender kernel inspect message certain type datum instance kernel process port information message translate port internal port datum structure address forward processing netmsgserver section d.5.3 use pointer message provide mean transfer entire address space task single message kernel process pointer line datum pointer datum sender address space invalid receiver’s especially sender receiver reside different system generally system send message copy datum sender receiver technique inefficient especially large message mach take different approach datum refer- pure type datum memory cache object memory cache object ence pointer message send port system copy sender receiver instead address map receive task modify include copy write copy page message operation fast datum copy make message passing efficient essence message passing implement virtual version 2.5 operation implement phase pointer region memory cause kernel map region space temporarily set sender memory map copy write mode ensure modification affect original version datum message receive destination kernel move mapping receiver address space newly allocate region virtual memory task version 3 process simplify kernel create data structure copy region address map receipt data structure add receiver map copy accessible receiver newly allocate region task need contiguous previous allocation mach virtual memory say sparse consist region datum separate unallocated address message transfer show figure d.6 message send computer message destination locate message transmit destination unix tra- ditionally leave mechanism low level network protocol require use statically assign communication endpoint example mach system mach message transfer port number service base tcp udp mach tenet object system location independent loca- tion transparent user tenet require mach provide location- transparent naming transport extend ipc multiple computer naming transport perform

transparent naming transport extend ipc multiple computer naming transport perform network message server netmsgserver user level capability base networking daemon for- ward message host provide primitive network wide service allow task register port lookup task computer network mach port transfer message message send port primitive service solve problem transfer port subsequent ipc interaction fully transparent netmsgserver track right line memory pass intercomputer message arrange appropriate transfer netmsgservers maintain distribute database port right transfer computer port right correspond kernel use netmsgserver message need send port kernel computer mach kernel ipc transfer message local netmsgserver netmsgserver use network protocol appropriate transfer message peer computer notion netmsgserver protocol independent netmsgservers build use protocol course netmsgservers involve transfer agree protocol finally netmsgserver destination computer use kernel ipc send message correct destination task ability extend local ipc transparently node support use proxy port send right transfer computer netmsgserver destination computer create new port proxy represent original port destination message send proxy receive netmsgserver forward transparently original port procedure example netmsgservers cooperate proxy indistinguishable original port mach design function network heterogeneous sys- tem provide way system send datum format way understandable sender receiver unfortunately computer differ format use store type datum instance integer system 2 byte store significant byte store significant system reverse ordering netmsgserver use type information store message translate datum sender receiver format way datum represent correctly reach destination netmsgserver give computer accept rpc add look remove network port netmsgserver service secu- rity precaution port value provide add request port match remove request thread ask port remove database example netmsgserver operation consider thread node send message port happen task node b. program simply send message port send right message pass kernel deliver recipient netmsgserver node a.

netmsgserver contact database information netmsgserver node b send message netmsgserver node b present message kernel appropriate local port node b.

kernel finally provide message receive task thread task execute msg receive sequence event show figure d.7 mach 3.0 provide alternative netmsgserver improve support norma multiprocessor norma ipc subsystem mach 3.0 implement functionality similar netmsgserver directly network ipc forwarding netmsgserver mach system mach kernel provide efficient internode ipc multicomputer fast interconnection hardware example time consume copying message netmsgserver kernel eliminate use norma ipc preclude use netmsgserver netmsgserver provide mach ipc service network link norma multiprocessor computer addition norma ipc mach 3.0 provide support memory management norma system enable task system create child task node feature support implementation single system image operate system norma multiprocessor multiprocessor behave like large system assemblage small system user application synchronization ipc ipc mechanism extremely flexible mach example thread synchronization port synchronization variable n message send n resource thread wish use resource execute receive port thread receive message resource available wait port message available return resource use thread send message port regard receiving equivalent semaphore operation wait sending equivalent signal method synchronize semaphore operation thread task synchronization task task receive right port general purpose semaphore simple daemon write implement method give object orient nature mach surprising principal abstraction mach memory object memory object manage secondary storage generally represent file pipe datum map virtual memory read write figure d.8 memory object back user level memory manager place traditional kernel incorporate virtual memory pager find operate system contrast traditional approach have kernel manage secondary storage mach treat secondary storage object usually file object system object port associate manipulate message send port memory object unlike memory management routine monolithic traditional kernel allow easy experimentation new memory manipulation algorithm virtual address space task generally sparse consist hole unallocated space instance memory map file place set address large message transfer share memory segment segment section virtual memory address provide thread access message

segment segment section virtual memory address provide thread access message new item map mach virtual memory task address map remove address space hole unallocated memory appear mach make attempt compress address space task fail crash room request region address space give address space 4 gb limitation currently problem maintain regular page table 4 gb address space task especially hole use excessive amount memory 1 mb key sparse address space page table space currently allocate region page fault occur kernel check page valid region sim- ply index page table check entry result lookup complex benefit reduce memory storage requirement simple address space maintenance approach worthwhile mach system call support standard virtual memory function- ality include allocation deallocation copying virtual memory allocate new virtual memory object thread provide address object let kernel choose address physical mem- ory allocate page object access object backing store manage default pager section d.6.2 virtual memory object mach system allocate automatically task receive message contain associated system call return information memory object task address space change access protection object specify object pass child task time creation shared copy write present user level memory manager secondary storage object usually map virtual address space task mach maintain cache memory resident page map object virtual memory implementation page fault occur thread access nonresident page execute message object port concept memory object create service nonkernel task unlike thread instance create maintain kernel important end result traditional sense memory page user write memory manager object destroy memory manager write change page secondary storage assumption mach content importance memory object memory object independent kernel circumstance user level memory manager insufficient instance task allocate new region virtual memory memory manager assign region represent secondary storage object page memory manager fail perform pageout mach need memory manager care memory need case mach provide default memory

memory manager care memory need case mach provide default memory manager mach 2.5 default memory manager use standard file system store datum write disk require separate swap space 4.3 bsd mach 3.0 osf/1 default memory manager capable file standard file system dedicated disk partition default memory manager interface similar user level one extension support role memory manager rely perform pageout user level manager fail pageout policy implement internal kernel thread pageout daemon paging algorithm base fifo second chance section 10.4.5 select page replacement select page send appropriate manager user level default actual pageout user- level manager intelligent default manager implement different paging algorithm suitable object back select page forcibly page user level manager fail reduce resident set page ask kernel default memory manager invoke page user level manager reduce user level manager resident set size user- level manager recover problem prevent perform pageout touch page cause kernel page page see fit thread need access datum memory object instance file invoke vm map system include system port identify object memory manager responsible region kernel execute call port datum read write region add complexity kernel make call asynchronously reasonable kernel wait user level thread unlike situation pageout kernel recourse request satisfy external memory manager kernel knowledge content object object memory manager responsible consistency content memory object map task different machine task single machine share single copy map memory object consider situation task different machine attempt modify page object time manager decide modification serialize conservative manager implement strict memory consistency force modification serialize grant- e write access kernel time sophisticated manager allow access proceed concurrently example manager know task modify distinct area page merge modification successfully future time external memory manager write mach example implement- e map file implement logic deal multiple kernel complexity logic vm map memory object kernel send message memory

complexity logic vm map memory object kernel send message memory manager port pass invoke mem- ory manager init routine memory manager provide support memory object port pass mem- ory manager control port port control port memory manager provide datum kernel example page resident port mach receive message simply point reference comparison finally memory object respond memory manager init memory object set attribute indicate ready accept request task send right memory object relinquish right kernel deallocate object port free memory manager memory object destruction kernel call need support external memory manager vm map discuss addition command set attribute provide page level locking require instance page fault occur memory manager return appropriate datum memory manager pass page multiple page read ahead kernel response page fault necessary kernel invoke memory manager asynchronously finally call allow memory manager report error kernel memory manager provide support call support object discuss memory object init thread cause page fault memory object page ker- nel send memory object datum request memory object port behalf fault thread thread place wait state mem- ory manager return page memory object datum provide return appropriate error kernel page mach system modify precious page kernel need remove resident memory page aging instance send memory object memory object datum write precious page page modify discard memory manager long retain copy memory man- ager declare page precious expect kernel return remove memory precious page save unnecessary dupli- cation copying memory current version mach allow external memory manager affect page replacement algorithm directly mach export memory access information need external task select recently page instance method provide informa- tion

currently investigation external memory manager useful variety reason reject kernel replacement victim know well candi- date instance mru page replacement monitor memory object back request page page memory usage invoke mach pageout daemon especially important maintain consistency secondary storage thread multiple processor section d.6.3 control order operation secondary storage enforce consistency constraint demand database management system example transaction logging transaction write log file disk modify database datum control map file access mach use share memory reduce complexity system facili- tie provide feature efficient manner shared memory generally provide extremely fast interprocess communication reduce over- head file management help support multiprocessing database management mach use share memory traditional share memory role instance thread task share task memory formal share memory facility need task mach provide traditional share memory support operate system construct unix fork system obviously difficult task multiple machine share memory maintain data consistency mach try solve problem directly provide facility allow problem solve mach support consistent share memory memory share task run processor share memory aparent task able declare region memory inherit child readable writable scheme different copy write inheritance task maintain copy change page writable object address task address map change copy thread task responsible coordinate change memory interfere write location concurrently coordination normal synchronization method critical section mutual exclusion lock case memory share separate machine mach allow use external memory manager set unrelated task wish share section memory task use external memory manager access secondary storage area implementor system need write task external pager pager simple complicated need simple implementation allow reader page write write attempt cause pager invalidate page task currently access pager allow write revalidate reader new version page reader simply wait page fault page available mach provide memory manager network memory server netmemserver multicomputer norma configuration mach 3.0 provide similar support standard kernel xmm

norma configuration mach 3.0 provide similar support standard kernel xmm subsystem allow multicomputer system use external memory manager incorporate logic deal multiple kernel xmm subsystem responsible maintain data consistency multiple kernel share memory make kernel appear single kernel memory manager xmm subsystem imple- ment virtual copy logic map object manage virtual copy logic include copy reference multicomputer kernel sophisticated copy write optimization programmer work level mach course system level mach 2.5 equivalent 4.3 bsd system interface version 2.5 include 4.3 bsd thread kernel bsd system trap kernel service thread behalf caller standard bsd handle emulation multithreaded limited efficiency mach 3.0 move single server model support multiple server true microkernel feature normally find kernel functionality provide emu- lation library server combination keep defini- tion microkernel emulation library server run outside kernel user level way multiple operate system run concurrently mach 3.0 kernel emulation library set routine live read program address space operating system call program make translate subroutine call library single user operate system ms dos macintosh operate system implement solely emulation library efficiency reason emulation library live address space program need functionality theory how- separate task complex operate system emulate use library server system call implement library redirect appropriate server server multithreade