Chapter 1: Intro

OS is a program that acts as an intermediary between a user of a computer and the computer hardware.

Goals: Execute user programs and make solving user problems easier; computer sys convenient to use; efficiently use hardware; hide硬件peculiarities. (Res allocator, ctl program)

Computer sys: Hardware (CPU, mem, I/O device; basic computing resources), OS (Ctl & coordinate use of hardware among apps and users), apps (compilers), user

Computer startup: bootstrap program (EPROM, firmware

CPU & I/O: execute concurrently; Each device controller has a **local buffer**; CPU moves data from/to main memory to/from local buffers; I/O is from the device to local buffer of controller; I/O interrupt.

Interrupt: transfer control to ISR (interrupt service routine); mark the addr of the interrupted instruction; interrupt vector contains the addr of all the service routines; OS is interrupt driven; interrupt request (IR)

Trap & Exception: software interrupt. (Interpt is hardware)

Type of interrupts: Polling(CPU check)&vectored interrupt sys

Asynchronous IO: Sys call, Device-status table (IO device's type, address, and state)

Mem-device hierarchy: main mem (RA, volatile), secondary storage (non-volatile), solid state disks, magnetic disk (non-v, tracks, sections)

Caching: Information in use copied from slower to faster storage temporarily. Cache size and replacement policy.

Device Driver: provides uniform interface between controller and kernel

Direct Mem Access: Only one interrupt is generated per block, rather than the one interrupt per byte

Processors: single general purpose processor; multiprocessors/ paralel system -> throughput, economy of scale, graceful degradation or fault tolerance. (sync, async)

Clustered Sys: Like multiprocessor systems, but multiple systems working together; storage area network; high-availability service which survives failures (sync, async clustering); HPC; Distributed Lock Manager (conflict ops)

Dual mode: kernel and user mode; mode bit; sys call change to kernel mode and return to user mode; VMM mode.

Process: Program is a passive entity, process is an active entity. A process is a program in execution. One thread, one PCounter

Process management: create&delete user&sys processes; suspending and resuming; p sync, p communicate; deadlock

Memory management: keep track of usage and by whom; move in and out; allocate and deallocate.

Storage Management: abstract physical properties to logical storage unit - file.

IO subsys: Memory management of I/O including buffering (storing data temporarily while it is being transferred), caching (storing parts of data in faster storage for performance), spooling (the overlapping of output of one job with input of other jobs); General&specific device-driver interface

Protection and Security: user id, group id, privilege escalation

Computing environment: traditional, mobile, distributed, client-server, peer-to-peer, virtualization, cloud computing, real-time embedded sys.

Chapter2: Sys Structure

OS services: UI (GUI, TUI, CLI, Batch), program exec, IO op, file-sys, communication, error detection, resrc allocation, accounting, protection and security.

Sys calls: programming interface to services provided by OS; in C/C++; accessed via API rather than direct use (hide details)

Passing parameters: 1)pass to reg 2)store in a block, table in mem, pass the block addr 3)onto the stack by program and pop by OS.

Sys programs: in user's view, OS is defined by system programs, not sys calls; file management; statues info; file modification; programming-lang-support; program-load and exec; communications; background services; app programs

OS design goals: 1) user:convenient to use, easy to learn, reliable, safe, and fast 2) sys:easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient.

Micro kernel: extend; port OS to new architectures; more reliable and secure. Downside: performance overhead

Loadable Kernel Modules: most popular OS implementatior

Dynamically loadable kernel: kernel extensions

Monolithic kernel:

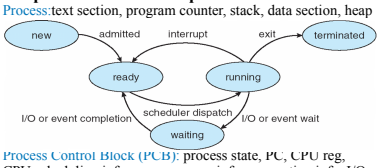
IOS: Cocoa Touch, Media Service, Core Service, Core OS

Kernighan's Law: "**Debugging** is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it." (log files; core dump; crash dump; tracing list; profiling; DTrace tool)

Sys Booting: bootstrap loader in ROM/EEROM locates the kernel; two step, boot block in ROM which then load the boot loader (commonly GRUB->allow kernel selection) from disk.

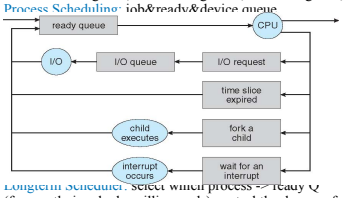
Chapter3: Process Concept

Process:text section, program counter, stack, data section, heap



Process Control Block (PCB): process state, PC, CPU reg, CPU scheduling info, mem-manage info, accounting info, I/O

Process Scheduling: job&ready&device queue

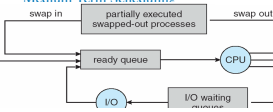


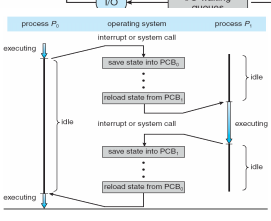
Longterm Scheduler: select which process -> ready Q (frequently invoked - milliseconds) control the degree of multi-programming

Shortterm Scheduler: select which process -> exec & CPU (infrequently invoked - seconds, minutes)

CPU-bound / IO-bound process: longterm sch finds good mix

Medium Term Scheduling:





Context Switch

Process Creation: parent create child which in turn create child. (Resource Sharing / Execution Options); The address space duplicated and child program is loaded into it. (fork/exec/wait)

Process Identifier: pid used to manage processes.

Process Termination: exit() -> ask the OS to delete it; abort() - parent may terminate the child

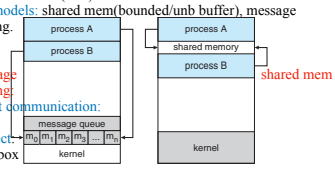
Cascading Termination: terminate all child when parent term

Zombie: a terminated child p without parent waiting

Orphans: If parent terminated, the process is an orphan.

Cooperating Process: supports info sharing, computation speedup, modularity, convenience -> needs interprocess communication (IPC).

IPC models: shared mem(bounded/unb buffer), message passing.



Message Passing: shared mem

Direct communication: or

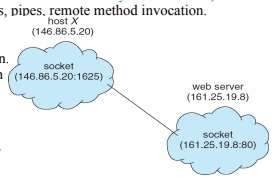
Indirect: Mail box

Synchronization: Message passing may be blocking (sync) (sender&receiver blocked until received or mesg available) or non-b (async) (sender send and continue, receiver may continue with null)

Buffering: zero capacity (sender must wait receiver), bounded capacity (finite len of mesg, sender wait if full), unbounded c.

Communication in Client-Server Systems: sockets, remote procedure calls, pipes, remote method invocation.

Socket: endpoint for communication. Concatenation of IP address and port.



Three types of Sockets:

Connection-oriented(TCP), Connectionless(UDP), MulticastSocket class.

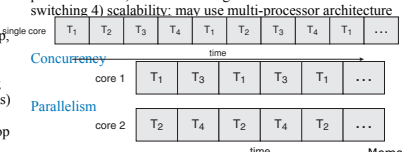
Remote Procedure Call: stubs (client side) locate the server and marshal the parameters, server-side stub receives this message, unpacks the marshalled parameters, and performs the procedure on the server

Chapter4: MultiThread Programming

Motivation: process is heavy weight; simplify code, increase efficiency; kernels are generally multithreaded.

Processes are completely separated program with its own variables, stacks and mem allocation while Threads shares the same mem space and global variables between routines.

Benefits: 1) responsiveness: May allow continued execution if part of process is blocked, especially important for user interfaces 2) resource sharing 3) economy: Cheaper than process creation, thread switching lower overhead than context switching 4) scalability: may use multi-processor architecture



Amdahl's Law: speedup <= 1 / (S + (1-S)/N)

where S - the serial portion, N - the processing cores.

User thread & Kernel thread: Many to One (Many user-level threads mapped to single kernel thread, one thread blocked causes all to block), One to One (more concurrent), Many to Many (OS create sufficient number of kernel threads),two level

Pthread: may be provided either as user level or kernel level

```
pthread_mutex_lock(&mutex1);
while (c <> 0)
  pthread_cond_wait(cond1,mutex1);
pthread_mutex_unlock(&mutex1);
take_action();
```

```
pthread_mutex_lock(&mutex1);
c--;
if (c == 0) pthread_cond_signal(cond1);
pthread_mutex_unlock(&mutex1);
.
```

Signals are not remembered

Implicit Threading: creation and management of threads done by compilers and runtime libraries rather than programmers (methods: Thread Pools, OpenMP, GrandCentral Dispatch)

Thread Pools: create a number of threads in a pool where they await work. Advantages: 1) slightly faster to service a request with an existing thread than create a new thread 2) # of threads in app be bound to the size of the pool 3) separate task from mechanics of create tasks-different strategies for running task

OpenMP: set of compiler derivatives and an API for C, C++, FORTRAN; Support for parallel programming in shared-memory environments; Identifies parallel regions – blocks of code that can run in parallel.

Grand Central Dispatch: Apple technology for Mac OS X and iOS operating systems. Similar to OpenMP's function.

Thread Issues: ***fork*** (duplicate all?)/exec; ***Signal Handling*** (handler routine; deliver to -> process; thread which the signal applies every thread; certain thread; assign a thread to take all); ***Thread Cancelation*** (async / deferred cancelation); Thread-***Local Storage*** (allows each thread to have its own copy of data; similar to static data); ***Scheduler Activities*** (appropriate number of kernel threads allocated to the application (1:1/m:1); use an intermediate data structure between user and kernel threads – lightweight process (LWP))

Chapter8: Memory Management

OS: Process/Mem/Storage(File)/IO management

Background: Program must be brought (from disk) into memory and placed within a process for it to be run; Main memory and registers are only storage CPU can access directly.

Base and Limit Reg: define the logical address space. CPU check every mem access from user mode to make sure it is between the base and limit for that user.

Addr Binding: Compile time (addr known at compile time - absolute code), Load time (must generate relocatable code), Execution time (allow process to be moved during its exec)

MultiStep Processing of a user program: scr - [compiler/ assembler] - object module - [linkage editor] (other object modules comes in) - load module - [loader] (sys lib comes in) - [in-mem binary mem image] (dynamic loaded sys lib in)

Logical & Physical addr: same in compile-time and load time addr-binding schema. Different in exec-time addr binding. User program deal with logical addr, it never sees physical addr. Execution-time binding occurs when reference is made to location in memory.

Mem management Unit: hardware device that at run time maps virtual to physical addr.

Dynamic relocation:

Static Linking: system libraries and program code combined by the loader into the binary program image

Dynamic Linking: linking postponed until execution time; use a piece of code (stab) to locate and replace itself with the appropriate memory-resident library routine. OS check if routine is in the addr space add if not. Particular useful to libs.

Swapping: A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution. Backing store: fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images.

Roll out Roll in: swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed. Depending on the binding method and IO pending, the process may be swapped back to the exact same addr or not.

Overhead proportional to the swap size.

Mordern OS: Swap only when free memory extremely low

Swapping on Mobile sys: Not typically supported, but has flash for quick restart.

Contiguous Allocation: OS usually held in low mem with interrupt vector. User process in high addr. Each process contained in a single contiguous mem section. Relocation registers (base, limit, relocation) used to protect user processes from each other, and from changing operating-system code and data.

Multiple Partition Allocation: limit the degree of multiprograming; Variable-partition size; Hole (fragmentation)

First fit / Best fit / Worst fit

External Fragmentation: total memory space exists to satisfy a request, but it is not contiguous (and when none of them is large enough to fit the program alone) -> compaction: Shuffle memory contents to place all free memory together in one large block.

Internal Fragmentation: size difference between allocated mem and used is memory internal to a partition, but not being used.

Segmentation: logical addr <segment num, offset>; vary length

Segmentation table: base contains the starting physical address where the segments reside in memory; limit specifies the length of the segment. Each entry is associated to validation bit

Segmentation base/length reg (STBR, STLR)

Paging: PM -> frames; LM -> pages; Page table to translate; p N pages process need N free frames to be loaded. Backing store likewise split into pages. Still has fragmentation.

Page Num & Page Offset

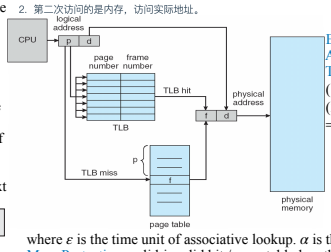
Trade-off: small page size, less fragmentation but larger page table

Page Table Implementation: Page table is kept in the mem.

Page table base / length reg: this scheme every data/instruction access requires two memory accesses. Translation look-aside buffers / Associative Reg (TLBs) are used for the two men access procedure. Address-space identifiers (ASIDs) may be stored in each entry.

1. 第一次访问内存是访问页表，取出虚拟页对应的物理页。
2. 第二次访问内存是访问实际内存地址。

1. 第一次访问TLB，得到虚拟页对应的物理页
2. 第二次访问的是内存，访问实际地址。



Effective Access Time: $(1+\varepsilon)*\alpha+(2+\varepsilon)*(1-\alpha) = 2+\varepsilon-\alpha$

where ε is the time unit of associative lookup. α is the hit ratio.

Mem Protection: valid-invalid bit / page table length reg.

Shared Pages: One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems)

Structure of Page Table: Using straight forward method the page table may become enormous

Memory structures for paging can get huge using straight-forward methods

Consider a 32-bit logical address space as on modern computers

Page size of 4 KB (2^{12})

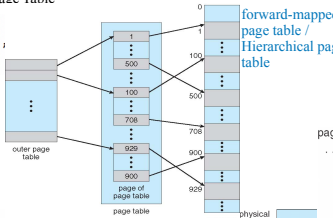
Page table would have 1 million entries (2^{32} / 2^{12})

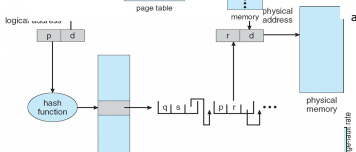
If each entry is 4 bytes -> 4 MB of physical address space / memory for page table alone

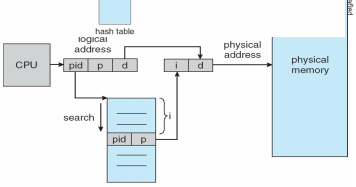
That amount of memory used to cost a lot

Don't want to allocate that contiguously in main memory

Solutions: Hierarchical Paging, Hased Page Tables, Inverted Page Table







Chapter9: Virtual Memory Management

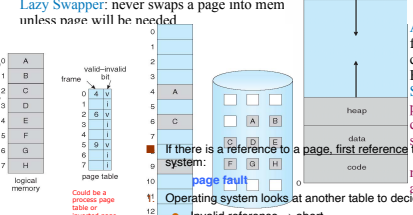
Background: All program need to be in men to exec, but entire program is not needed at the same time.

Virtual Mem: separation of user logical memory from physical memory; Adv: logical >> physical; addr space sharing; efficient process creation; concurrency; Less IO (page<process

Implementation: Demand Paging, Demand Segmentation.

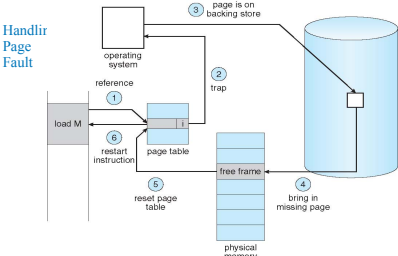
Virtual Addr Space: Enables sparse address spaces with holes left for growth, dynamically linked libraries, etc.

Lazy Swapper: never swaps a page into mem unless page will be needed



Page fault:

1. If there is a reference to a page, first reference to that page will trap to operating system: **page fault**
2. Operating system looks at another table to decide:
   - Invalid reference ⇒ abort
   - Just not in memory
3. Get empty frame
4. Swap page into frame via scheduled disk operation
5. Reset tables to indicate page now in memory
   Set validation bit = **v**
6. Restart the instruction that caused the page fault



Locality of reference: result in less pain in -> a given instruction could access multiple pages -> multiple page faults

Instruction Restart:

Demand Paging:

Stages in Demand Paging

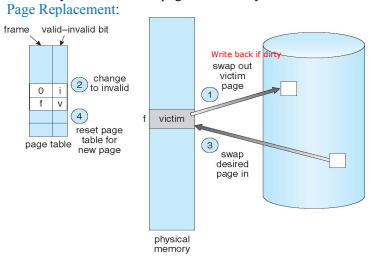
1. Trap to the operating system
2. Save the user registers and process state
3. Determine that the interrupt was a page fault
4. Check that the page reference was legal and determine the location of the page on the disk
5. Issue a read from the disk to a free frame:
   1. Wait in a queue for this device until the read request is serviced
   2. Wait for the device seek and/or latency time
   3. Begin the transfer of the page to a free frame
6. While waiting, allocate the CPU to some other user
7. Receive an interrupt from the disk I/O subsystem (I/O completed)
8. Save the registers and process state for the other user
9. Determine that the interrupt was from the disk
10. Correct the page table and other tables to show page is now in memory
11. Wait for the CPU to be allocated to this process again
12. Restore the user registers, process state, and new page table, and then resume the interrupted instruction

Effective Access Time (EAT): (1-p) * mem access + p * (page fault overhead + swap page out + in + restart overhead)

where p is the page fault rate.

Copy on write (COW): allows both parent and child processes to initially share the same pages in memory.

Page Replacement:



Page and Frame replace algorithm: 1) frame allocation: # of frames to each process 2) Page Replacement algo: FIFO, LRU (Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter); second chance also (reference bit, generally FIFO: Reference bit = 0 -> replace it, reference bit = 1 then: set to zero and go to next)

Counting Algo: LFU / MFU

Page Buffering: Maintain a frame pool

Fixed / Proportion / Priority allocation: this are frame allocation algorithms

Global allocation: process selects a replacement frame from the set of all frames; one process can take a frame from another

Local Allocation: each process selects from only its own set of allocated frames

Non-Uniform Mem Access: Optimal performance comes from allocating memory "close to" the CPU on which the thread is scheduled (Igroups: struct to track low latency CPU/ Mem groups)

Thrashing: a process is busy swapping pages in and out; cause low CPU utilization, OS think it should increase the degree of multiprogramming, add processes to the sys.

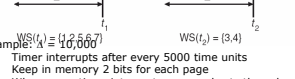
Why demand paging works: Locality model

Why thrashing occurs: sum of locality > total mem size

Working Set Model: WSSi (working set of process Pi) = total number of pages referenced in the most recent Δ (varies in time); D = $\sum$WSS$_i$ = total demand frames; if D > mem(thrash

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 3 2 3 4 4 4 3 4 4 4 . . .



WS(t_1) = {1,2,5,6,7}

WS(t_2) = {3,4}

ample: Δ = 10,000

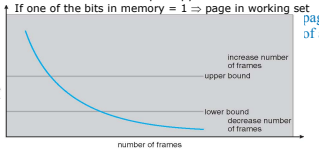
Timer interrupts after every 5000 time units

Keep in memory 2 bits for each page

Whenever a timer interrupts copy and sets the values of all reference bits to 0

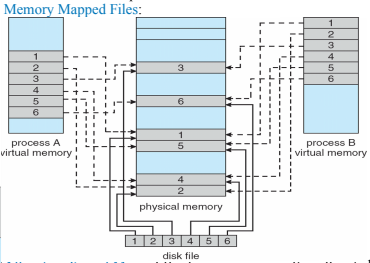
If one of the bits in memory = 1 ⇒ page in working set

Page fault frequency of a process



Above are two access patterns to the pages one in terns of replacing and actual allocation (of pages in or across all mem), the other is in terms of determining the total number of frames to be allocated to the process.

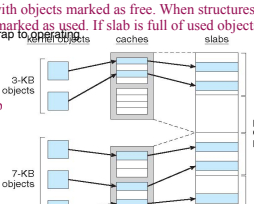
Memory Mapped Files:



Allocating Kernel Mem: differ from user mem, often allocated from a free-mem pool. (Various size, some need to be continuous)

Buddy sys:

Slab Allocator: slab is one or more physically contiguous pages. Cache consists of one or more slabs. When cache created, filled with objects marked as free. When structures stored, objects marked as used. If slab is full of used objects, next object allocated from empty slab. If no empty slabs, new slab allocated.



Pre-paging: To reduce the large number of page faults that occurs at process startup.
Other issues: Page Size (fragmentation, page table size, resolution, IO overhead, # of page faults, Locality, TLB size and effectiveness), TLB reach, Program struct, IO interlock (frames used by IO must be in men)

Chapter10: File Management

File: an abstract data type, a contiguous logical addr space. (text, source, executable…) (program / data)
File attributes: name (only info in human-readable), identifier (unique tag identifying file in the file sys), type, location (ptr to file location on device), size, protection, time, date, and user identification.
File Directory: Information about files are kept in the directory structure, which is maintained on the disk.
File Operations: create, write, read, seek (reposition within file), delete, truncate, open(F_i), close(F_i)
Open File manage: open-file table(track opened files), file pointer (to last read/write position), file-open count (num of time a file been opened allow removal of data from open-file table when last processes closes it), disk location, access right.
Open File Lock: shared lock — read lock (process concurrently); exclusive lock — write lock. Mandatory (强制加锁机制): access is denied depending on locks held and requested. Advisory (建议加锁机制): processes can find status of locks and decide what to do.

executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked

File Structure:OS/Program决定none(sequence of words, bytes), simple record struct(lines, variable/fixed length), complex struct.
Access Method: access file into when load into m…
Sequential-access file:
读写自动移动文件指针到下一个同时可以返回文件开头

sequential access	implementation for direct access
reset	cp = 0;
read next	read cp; cp = cp + 1;
write next	write cp; cp = cp + 1;

Direct Access:file is fixed Len… Logical records. read n, write n… position to n,rewrite n, n = relative block number
For direct access, the file is viewed as a numbered sequence of block or record. (Disk allow random access)
Directory Structure: collection of nodes contain info of all files
Disk structure: can be divided into *partitions/minidisks/slices*. Can be protected by RAID. Can be used as *raw*(without file sys). Entity containing file sys is called volume. Each volume's file sys tracks file sys info *device directory*.
General purpose file sys & Special purpose file sys
Directory Ops: Search F, create F, Delete F, List a D, Rename F, Traverse F sys
Single level directory: single dir for all users
Two level dir:
Tree struct dir:
Acyclic Graph Dir: shared subdir and files
on delete: Backpointers (Daisy chain), entry-hold-count (Count=0 then del)
General Graph Dir: 1) allow only links to file not subdir 2)Garbage collectio 3) Cycle detection.
File sys mounting: A file sys must be mounted before it can be used. (Mount point, eg flash disk)
Share Files: sharing of file on multiuser sys. -> protection scheme (user ID, group ID)
Failure Mode: corruption of dir struct, (network failure, server failure in remote file sys)

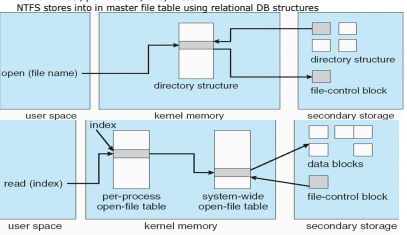
Chapter11: Implementing File Systems

File sys struct:
File struct: logical storage unit; collection of related info. File sys (on secondary storage): provide UI to storage, map logical to physical, provides efficient and convenient access to disk.
Disk (provide in place read and write and random access): IO transfers performed in blocks of sectors (group of sectors is a partition). File Control Block (FCB): storage structure consisting of information about a file. Device Driver: controls physical device
Layered File Sys: app -> logical file sys -> file organization module -> basic file sys -> IO control -> device.
Device drivers manage I/O devices at the I/O control layer
- Given commands like "read drive1, cylinder 72, track 2, sector 10, into memory locatio 1060" outputs low-level hardware specific commands to hardware controller
Basic file system given command like "retrieve block 123" translates to device driver
- Also manages memory buffers and caches (allocation, freeing, replacement)
  - Buffers hold data in transit
  - Caches hold frequently used data

File organization module understands files, logical address, and physical blocks
- Translates logical block # to physical block #
- Manages free space, disk allocation

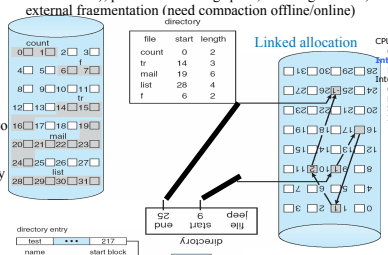
Logical file system manages metadata information
- Translates file name into file number, file handle, location by maintaining file control blocks (inodes in Unix)
- Directory management
- Protection

implementation - on disk and in mem structs
Boot control block contains info needed by system to boot OS from that volume
Needed if volume contains OS, usually first block of volume
Volume control block (superblock, master file table) contains volume details
Total # of blocks, # of free blocks, block size, free block pointers or array
Directory structure organizes the files
Names and inode numbers, master file table
Per-file File Control Block (FCB) contains many details about the file
Inode number, permissions, size, dates
NTFS stores into in master file table using relational DB structures

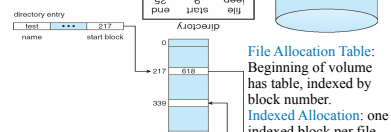


Partitions: can be a volume containing a file system ("cooked") or raw – just a sequence of blocks with no file system.
Boot block: can point to boot volume or boot loader set of blocks that contain enough code to know how to load the kernel from the file system. usually at the beginning (first block on first track) of a storage medium that holds special data used to start a system
Virtual File Sys:
Directory Implementation: Linear list (time consuming), B+ tree, Hash (linear list with hash table struct) (good if entries are fixed size/use chained-overflow method)
Allocation Method: how disk blocks are allocated for files.
Contiguous, Linked, Indexed

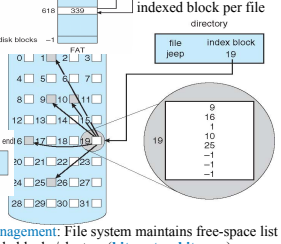
Contiguous Allocation: each file occupies set of contiguous blocks; best performance in most cases; simple (start & length are recorded); problems - finding space, knowing file size, external fragmentation (need compaction offline/online)

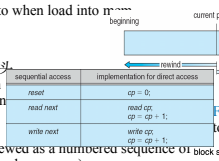


Linked allocation

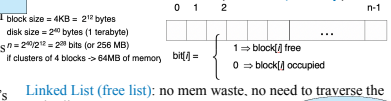


File Allocation Table: Beginning of volume has table, indexed by block number.
Indexed Allocation: one indexed block per file

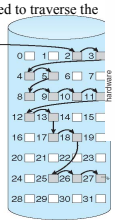




Free Space Management: File system maintains free-space list to track available blocks/clusters (**bit vector, bit map**).



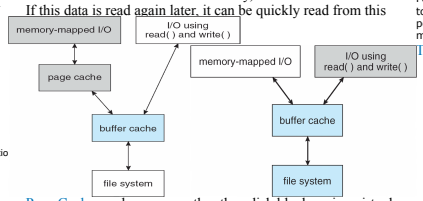
Linked List (free list): no mem waste. no need to traverse the entire list
Efficiency dependent on Disk allocation and directory algorithms
Types of data kept in file's directory entry
Pre-allocation or as-needed allocation of metadata structures
Fixed-size or varying-size data structures
Grouping
Modify linked list to store address of next *n-1* free blocks in the first free block, plus a pointer to next block that contains free-block-pointers (like this one)
Counting: addr of first free block, num of following free blocks.



Performance: 1) keeping data and metadata close together 2) buffer cache: separate section of main mem for frequently used blocks 3) free-behind, read ahead 4) sync / async write

1、Buffer（缓冲区）是系统两端处理速度平衡（从长时间尺度上看）时使用的。它的引入是为了减小短期内突发I/O的影响，起到流量整形的作用。比如生产者——消费者问题，他们产生和消耗资源的速度大体接近，加一个buffer可以抵消掉资源刚产生/消耗时的突然变化。
2、Cache（缓存）则是系统两端处理速度不匹配时的一种折衷策略。因为CPU和memory之间的速度差异越来越大，所以人们充分利用数据的局部性（locality）特征，通过使用存储系统分级（memory hierarchy）的策略来减小这种差异带来的影响。
3、假定以后存储器访问变得跟CPU做计算一样快，cache就可以消失，但是buffer依然存在。比如从网络上下载东西，瞬时速率可能会有较大变化，但从长期来看却是稳定的，这样就能通过引入一个buffer使得OS接收数据的速率更稳定，进一步减少对磁盘的伤害。
4、TLB（Translation Lookaside Buffer，翻译后备缓冲器）名字起错了，其实它是一个cache.

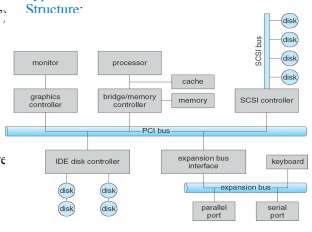
Mem-mapped I/O: Memory-mapped I/O uses the same address space to address both memory and I/O devices. The memory and registers of the I/O devices are mapped to (associated with) address values. So when an address is accessed by the CPU, it may refer to a portion of physical RAM, or it can instead refer to memory of the I/O device. Especially for large address spaces (graphics)
Page Cache/Buffer Cache: accelerates many accesses to files on non volatile storage. This happens because, when it first reads from or writes to data media like hard drives, Linux also stores data in unused areas of memory, which acts as a cache. If this data is read again later, it can be quickly read from this



Page Cache: caches pages rather than disk blocks using virtual memory techniques and addresses.
Unified buffer cache: uses the same page cache to cache both memory-mapped pages and ordinary file system I/O to avoid double caching
Recovery: consistent checking, backup (to other storage), restoring.
Log Structured/journaling File Systems: all transections are written into a log. The written one is considered committed, even if the file sys may not be updated yet.

Chapter13: I/O Systems

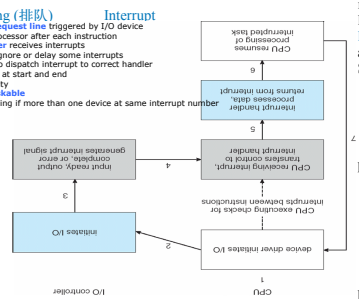
Ports (connection point for device), Buses (daisy chain or shared direct access), Device Controllers connect to diff devices. Driver (on OS). MEM_Mapped IO (instruction in mem)
Controller (host adapter): electronic that operate port, bus, device. Sometimes integrated, sometimes separate circuit board. Contains processor, microcode, private mem, bus controller.
Port: An I/O port is a socket on a computer that a cable is plugged into. The port connects the CPU to a peripheral device via a hardware interface or to the network via a network interface.
IO instructions control devices. Devices usually have registers where device driver places commands, addr, data.
Device Driver: encapsulate device details. Present uniform device-access interface to IO subsystem.
Variety of IO devices: storage, transmission, human-interface
Typical PC BUS Structure:



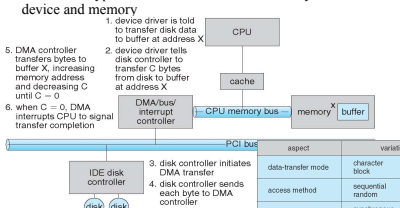
I/O address range (hexadecimal)	device
000–00F	DMA controller
020–021	interrupt controller
040–043	timer
200–20F	game controller
2F8–2FF	serial port (secondary)
320–32F	hard-disk controller
378–37F	parallel port
3D0–3DF	graphics controller
3F0–3F7	diskette-drive controller
3F8–3FF	serial port (primary)

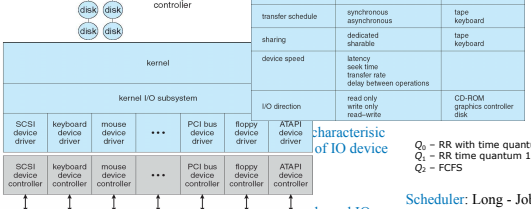
For each byte of I/O
1. Read busy bit from status register until 0 (busy wait)
2. Host sets read or write bit and if write copies data into data-out register
3. Host sets command-ready bit
4. Controller sets busy bit, executes transfer
5. Controller clears busy bit, error bit, command-ready bit when transfer done

Polling (排队) Interrupt
CPU **Interrupt-request line** triggered by I/O device
Checked by processor after each instruction
Interrupt handler receives interrupts
Maskable to ignore or delay some interrupts
Interrupt vector to dispatch interrupt to correct handler
Context switch at start and end
Based on priority
Some **nonmaskable**
Interrupt chaining if more than one device at same interrupt number



Direct Mem Access (DMA): used to avoid programmed IO (*one byte at a time*) for large data movement. Requests DMA controller. Bypasses CPU to transfer data directly between I/O device and memory

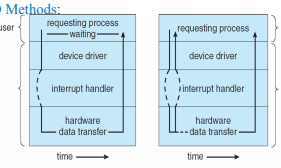




aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated shareable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read-write	CD-ROM graphics controller disk

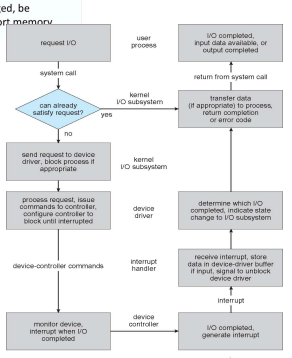
Characterisic of IO device
kernel IO Structure

Device controller, ioctl(); Types of IO devices: Block IO, Character IO (stream), men-mapped file access, network sockets.
Block device: commend -> read, write, seek; raw IO, direct access, file-sys access; mem-mapped file access; DMA.
Character device: get(), put(); keyboard, mice, serial ports; libs layered on top to allow line editing.
Network Device: Varying enough from block and character to have own interface. Approaches vary widely (pipes, FIFOs, streams, queues, mailboxes)
Clock and Timers: Provide current time当前, elapsed逝去时间, timer触发器. Normal resolution about 1/60 second. Programmable interval timer used for timings, periodic interrupts
Blocking IO: process suspended until I/O completed
Non-Blocking IO: I/O call returns (when available) as much as available. Implemented by multithreading.
Asynchronous IO: in most cases, same as non-blocking; return immediately, not waiting for the IO; 不一定对someIO作出反应
Well blocking IO means that a given thread cannot do anything more until the IO is fully received (in the case of sockets this wait could be a long time).
Non-blocking IO means an IO request is queued straight away and the function returns. The actual IO is then processed at some later point by the kernel.
For blocking IO you either need to accept that you are going to wait for every IO request or you are going to need to fire off a thread per request (Which will get very complicated very quickly).
For non-blocking IO you can send off multiple requests but you need to bear in mind that the data will not be available until some "later" point. This checking that the data has actually arrived is probably the most complicated part.
Two IO Methods:



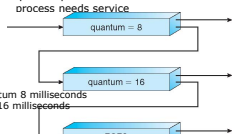
Kernel IO Subsystem:
Scheduling: I/O request ordering via per-device queue; fairness; implement Quality Of Service, device status table
Buffering: store data in memory while transferring between devices. 1) cope with device speed mismatch 2) cope with device transfer size mismatch 3) maintain "copy semantics" 支持应用IO的复制语义
Double Buffering: two copies of data. 1) kernel and user 2) varying size 3) full / being processed and not-full / being used 4) considers cop-on-write for efficiency
Cache: fast device holding copy of data保存数据副本高速缓存器
Spooling: hold output for a device保存设备输出的缓冲区
Device Reservation: provide exclusive access to a device; sys call for allocation and dealloc; watch out deadlock
Error Handling: Most return an error number or code when I/O request fails; System error logs hold problem reports
IO protection: 通过发出非法 IO 指令, 用户程序可以有意无意中断系统的正常操作; all IO instructions defined to be privileged, be performed via system calls; Memory-mapped and IO port memory locations be protected

Life cycle of An IO request



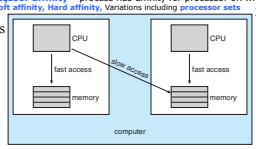
Chapter5: Process Scheduling

Dispatcher (分配器): Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves: switch context, switch to user mode, jump to proper location of user program to restart the program. Dispatch latency: Time for the dispatcher to stop one process and start another running.
Scheduling Criteria: CPU Utilization (keep cpu busy), Throughput (complete per time unit), Turnaround (time to exec a particular process) time, waiting time (time in the ready q), response time (time in waiting q)
FCFS: convoy effect: short process behind long process
SJF: how to know the span of next job/CPU burst?
Ask user or
1. t_n = actual length of n^{th} CPU burst
2. τ_{n+1} = predicted value for the next CPU burst
3. $\alpha, 0 \le \alpha \le 1$
4. Define :
Priority Scheduling: A priority number (integer) is associated with each process. Starvation (low priority processes may never execute) and Aging (as time progresses increase the priority of the process)
Round Robin: Each process gets a small unit of CPU time (time quantum q), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue. q + context switch.
Multilevel Queue: diff process apply diff scheduling algo and use diff queue. Fore-ground: RR, Back-ground: FCFS.
Scheduling must be done between the queues
Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR
20% to background in FCFS
Multilevel Feedback Queue: A process can move between the various queues; aging can be implemented this way.
MFQ can be define in the following way
Number of queues
Scheduling algorithms for each queue
Method used to determine when to upgrade a process
Method used to determine when to demote a process
Method used to determine which queue a process will enter when that process needs service

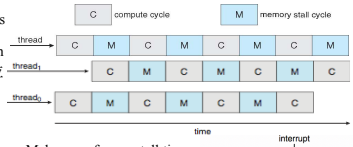


Q_0 – RR with time quantum 8 milliseconds
Q_1 – RR time quantum 16 milliseconds
Q_2 – FCFS

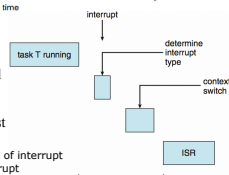
Scheduler: Long - Job scheduler -> Disk to Mem; Short - CPU scheduler -> Mem to CPU; Median - between Mem and Disk (ready但长时间不用，暂放回disk)
Multiprocessor Scheduling:
CPU scheduling more complex when multiple CPUs are available
Homogeneous processors within a multiprocessor
Asymmetric multiprocessing – only one processor accesses the system data structures, alleviating the need for data sharing
Symmetric multiprocessing (**SMP**) – each processor is self-scheduling, all processes in common ready queue, or each has its own private queue of ready processes(Currently, most common)
Processor affinity – process has affinity for processor on which it is currently running
Soft affinity, Hard affinity, Variations including **processor sets** 亲和性



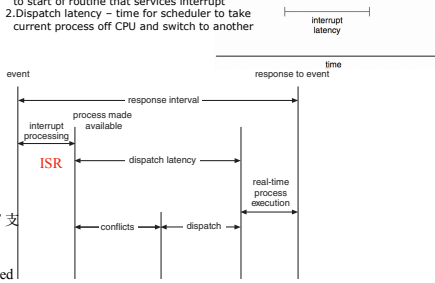
Load Balancing: attempts to keep workload evenly distributed.
Push migration: give put; Pull migration: take in



Make use of mem stall time.



Soft real-time sys: no guarantee as to when critical real-time process will be scheduled.
Hard real-time sys: task must be serviced by its deadline.
1.Interrupt latency – time from arrival of interrupt to start of routine that services interrupt
2.Dispatch latency – time for scheduler to take current process off CPU and switch to another



Rate Monotonic Scheduling: A priority is assigned based on the inverse of its period