

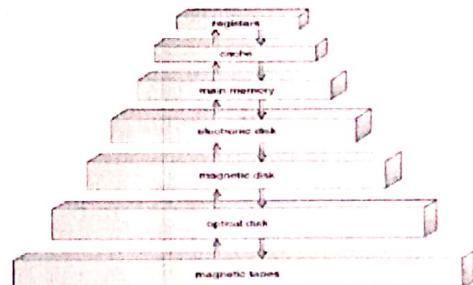
Study Guide to Accompany *Operating Systems Concepts* 9th Ed by Silberschatz, Galvin and Gagne

By Andrew DeNicola, BU ECE Class of 2012

Figures Copyright © John Wiley & Sons 2012

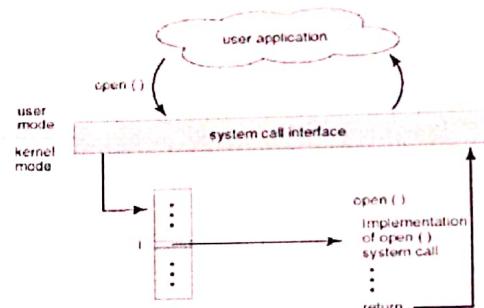
Ch.1 - Introduction

- An OS is a program that acts as an intermediary between a user of a computer and the computer hardware
- Goals: Execute user programs, make the comp. system easy to use, utilize hardware efficiently
- Computer system: Hardware ↔ OS ↔ Applications ↔ Users (\leftrightarrow = 'uses')
- OS is:
 - Resource allocator: decides between conflicting requests for efficient and fair resource use
 - Control program: controls execution of programs to prevent errors and improper use of computer
- Kernel: the one program running at all times on the computer
- Bootstrap program: loaded at power-up or reboot
 - Stored in ROM or EPROM (known as firmware), initializes all aspects of system, loads OS kernel and starts execution
- I/O and CPU can execute concurrently
- Device controllers inform CPU that it is finished w/ operation by causing an interrupt
 - Interrupt transfers control to the interrupt service routine generally, through the interrupt vector, which contains the addresses of all the service routines
 - Incoming interrupts are disabled while another interrupt is being processed
 - Trap is a software generated interrupt caused by error or user request
 - OS determines which type of interrupt has occurred by polling or the vectored interrupt system
- System call: request to the operating system to allow user to wait for I/O completion
- Device-status table: contains entry for each I/O device indicating its type, address, and state
 - OS indexes into the I/O device table to determine device status and to modify the table entry to include interrupt
- Storage structure:
 - Main memory – random access, volatile
 - Secondary storage – extension of main memory That provides large non-volatile storage
 - Disk – divided into tracks which are subdivided into sectors. Disk controller determines logical interaction between the device and the computer.
- Caching – copying information into faster storage system
- Multiprocessor Systems: Increased throughput, economy of scale, increased reliability
 - Can be asymmetric or symmetric
 - Clustered systems – Linked multiprocessor systems
- Multiprogramming – Provides efficiency via job scheduling
 - When OS has to wait (ex: for I/O), switches to another job
- Timesharing – CPU switches jobs so frequently that each user can interact with each job while it is running (interactive computing)
- Dual-mode operation allows OS to protect itself and other system components – User mode and kernel mode
 - Some instructions are only executable in kernel mode, these are privileged
- Single-threaded processes have one program counter, multi-threaded processes have one PC per thread
- Protection – mechanism for controlling access of processes or users to resources defined by the OS
- Security – defense of a system against attacks
- User IDs (UID), one per user, and Group IDs, determine which users and groups of users have which privileges



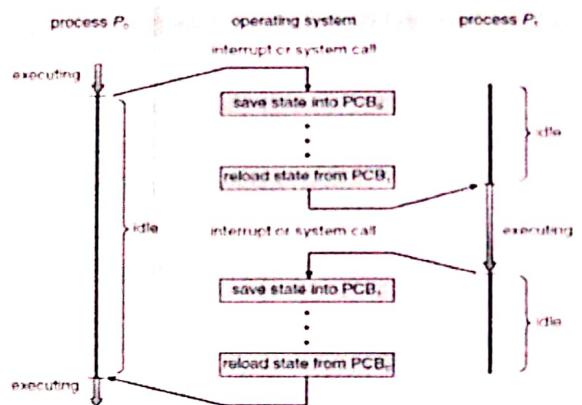
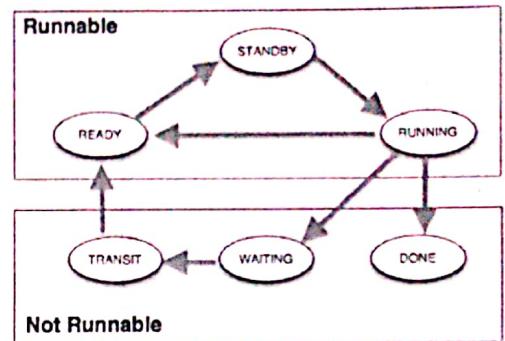
Ch.2 – OS Structures

- User Interface (UI) – Can be Command-Line (CLI) or Graphics User Interface (GUI) or Batch
 - These allow for the user to interact with the system services via system calls (typically written in C/C++)
- Other system services that are helpful to the user include: program execution, I/O operations, file-system manipulation, communications, and error detection
- Services that exist to ensure efficient OS operation are: resource allocation, accounting, protection and security
- Most system calls are accessed by Application Program Interface (API) such as Win32, POSIX, Java
- Usually there is a number associated with each system call
 - System call interface maintains a table indexed according to these numbers
- Parameters may need to be passed to the OS during a system call, may be done by:
 - Passing in registers, address of parameter stored in a block, pushed onto the stack by the program and popped off by the OS
 - Block and stack methods do not limit the number or length of parameters being passed
- Process control system calls include: end, abort, load, execute, create/terminate process, wait, allocate/free memory
- File management system calls include: create/delete file, open/close file, read, write, get/set attributes
- Device management system calls: request/release device, read, write, logically attach/detach devices
- Information maintenance system calls: get/set time, get/set system data, get/set process/file/device attributes
- Communications system calls: create/delete communication connection, send/receive, transfer status information
- OS Layered approach:
 - The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface
 - With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers
- Virtual machine: uses layered approach, treats hardware and the OS kernel as though they were all hardware.
 - Host creates the illusion that a process has its own processor and own virtual memory
 - Each guest provided with a 'virtual' copy of the underlying computer
- Application failures can generate core dump file capturing memory of the process
- Operating system failure can generate crash dump file containing kernel memory



Ch.3 – Processes

- Process contains a program counter, stack, and data section.
 - Text section: program code itself
 - Stack: temporary data (function parameters, return addresses, local variables)
 - Data section: global variables
 - Heap: contains memory dynamically allocated during run-time
- Process Control Block (PCB): contains information associated with each process: process state, PC, CPU registers, scheduling information, accounting information, I/O status information
- Types of processes:
 - I/O Bound: spends more time doing I/O than computations, many short CPU bursts
 - CPU Bound: spends more time doing computations, few very long CPU bursts
- When CPU switches to another process, the system must save the state of the old process (to PCB) and load the saved state (from PCB) for the new process via a context switch
 - Time of a context switch is dependent on hardware
- Parent processes create children processes (form a tree)
 - PID allows for process management
 - Parents and children can share all/some/none resources
 - Parents can execute concurrently with children or wait until children terminate
 - fork() system call creates new process
 - exec() system call used after a fork to replace the processes' memory space with a new program
- Cooperating processes need interprocess communication (IPC): shared memory or message passing
- Message passing may be blocking or non-blocking
 - Blocking is considered synchronous
 - Blocking send has the sender block until the message is received
 - Blocking receive has the receiver block until a message is available
 - Non-blocking is considered asynchronous
 - Non-blocking send has the sender send the message and continue
 - Non-blocking receive has the receiver receive a valid message or null



Ch.4 – Threads

- Threads are fundamental unit of CPU utilization that forms the basis of multi-threaded computer systems
- Process creation is heavy-weight while thread creation is light-weight
 - Can simplify code and increase efficiency
- Kernels are generally multi-threaded
- Multi-threading models include: Many-to-One, One-to-One, Many-to-Many
 - Many-to-One: Many user-level threads mapped to single kernel thread
 - One-to-One: Each user-level thread maps to kernel thread
 - Many-to-Many: Many user-level threads mapped to many kernel threads
- Thread library provides programmer with API for creating and managing threads
- Issues include: thread cancellation, signal handling (synchronous/asynchronous), handling thread-specific data, and scheduler activations.
 - Cancellation:
 - Asynchronous cancellation terminates the target thread immediately
 - Deferred cancellation allows the target thread to periodically check if it should be canceled
 - Signal handler processes signals generated by a particular event, delivered to a process, handled
 - Scheduler activations provide upcalls – a communication mechanism from the kernel to the thread library.
 - Allows application to maintain the correct number of kernel threads

Ch.5 – Process Synchronization

- Race Condition: several processes access and manipulate the same data concurrently, outcome depends on which order each access takes place.
- Each process has critical section of code, where it is manipulating data
 - To solve critical section problem each process must ask permission to enter critical section in entry section, follow critical section with exit section and then execute the remainder section
 - Especially difficult to solve this problem in preemptive kernels
- Peterson's Solution: solution for two processes
 - Two processes share two variables: int **turn** and Boolean **flag [2]**
 - **turn**: whose turn it is to enter the critical section
 - **flag**: indication of whether or not a process is ready to enter critical section
 - **flag[i] = true** indicates that process P_i is ready
 - Algorithm for process P_i :

```
do {
    flag[i] = TRUE;
    turn = j;
    while (flag[j] && turn == j)
        critical section
    flag[i] = FALSE;
    remainder section
} while (TRUE);
```
- Modern machines provide atomic hardware instructions: Atomic = non-interruptable
- Solution using Locks:

```
do {
    acquire lock
    critical section
    release lock
    remainder section
} while (TRUE);
```

- Solution using Test-And-Set: Shared boolean variable lock, initialized to FALSE

```
boolean TestAndSet (boolean *target){
    boolean rv = *target;
    *target = TRUE;
    return rv;
}

do {
    while ( TestAndSet (&lock) ) ; // do
    nothing
    // critical section
    lock = FALSE;
    // remainder section
} while (TRUE);
```

- Solution using Swap: Shared bool variable lock initialized to FALSE; Each process has local bool variable key

```
void Swap (boolean *a, boolean *b){
    boolean temp = *a;
    *a = *b;
    *b = temp;
}

do {
    key = TRUE;
    while ( key == TRUE )
        Swap (&lock,
&key );
    // critical section
    lock = FALSE;
    // remainder section
} while (TRUE);
```

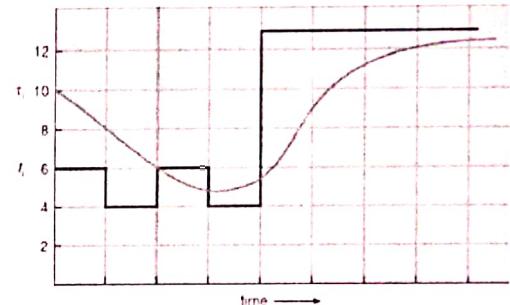
- Semaphore: Synchronization tool that does not require busy waiting
 - Standard operations: wait() and signal() ← these are the only operations that can access semaphore S
 - Can have counting (unrestricted range) and binary (0 or 1) semaphores
- Deadlock: Two or more processes are waiting indefinitely for an event that can be caused by only one of the waiting processes (most OSes do not prevent or deal with deadlocks)
 - Can cause starvation and priority inversion (lower priority process holds lock needed by higher-priority process)

Ch.5 – Process Synchronization Continued

- Other synchronization problems include Bounded-Buffer Problem and Readers-Writers Problem
- Monitor is a high-level abstraction that provides a convenient and effective mechanism for process synchronization
 - Only one process may be active within the monitor at a time
 - Can utilize condition variables to suspend a resume processes (ex: condition x, y;)
 - x.wait() – a process that invokes the operation is suspended until x.signal()
 - x.signal() – resumes one of processes (if any) that invoked x.wait()
 - Can be implemented with semaphores

Ch.6 – CPU Scheduling

- Process execution consists of a cycle of CPU execution and I/O wait
- CPU scheduling decisions take place when a process:
 - Switches from running to waiting (nonpreemptive)
 - Switches from running to ready (preemptive)
 - Switches from waiting to ready (preemptive)
 - Terminates (nonpreemptive)
- The dispatcher module gives control of the CPU to the process selected by the short-term scheduler
 - Dispatch latency- the time it takes for the dispatcher to stop one process and start another
- Scheduling algorithms are chosen based on optimization criteria (ex: throughput, turnaround time, etc.)
 - FCFS, SJF, Shortest-Remaining-Time-First (preemptive SJF), Round Robin, Priority
- Determining length of next CPU burst: Exponential Averaging:
 1. t_n = actual length of n^{th} CPU burst
 2. τ_{n+1} = predicted value for the next CPU burst
 3. $\alpha, 0 \leq \alpha \leq 1$ (commonly α set to 1/2)
 4. Define: $\tau_{n+1} = \alpha * t_n + (1-\alpha)\tau_n$
- Priority Scheduling can result in starvation, which can be solved by aging a process (as time progresses, increase the priority)
- In Round Robin, small time quantum can result in large amounts of context switches
 - Time quantum should be chosen so that 80% of processes have shorter burst times than the time quantum
- Multilevel Queues and Multilevel Feedback Queues have multiple process queues that have different priority levels
 - In the Feedback queue, priority is not fixed → Processes can be promoted and demoted to different queues
 - Feedback queues can have different scheduling algorithms at different levels
- Multiprocessor Scheduling is done in several different ways:
 - Asymmetric multiprocessing: only one processor accesses system data structures → no need to data share
 - Symmetric multiprocessing: each processor is self-scheduling (currently the most common method)
 - Processor affinity: a process running on one processor is more likely to continue to run on the same processor (so that the processor's memory still contains data specific to that specific process)
- Little's Formula can help determine average wait time per process in any scheduling algorithm:
 - $n = \lambda \times W$
 - $n = \text{avg queue length}; W = \text{avg waiting time in queue}; \lambda = \text{average arrival rate into queue}$
- Simulations are programmed models of a computer system with variable clocks
 - Used to gather statistics indicating algorithm performance
 - Running simulations is more accurate than queuing models (like Little's Law)
 - Although more accurate, high cost and high risk



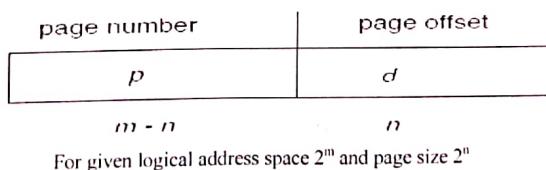
CPU burst (t)	6	4	6	6	4	13	13	13	...
"guess" (t):	10	8	6	5	5	9	11	12	...

Ch.7 – Deadlocks

- Deadlock Characteristics: deadlock can occur if these conditions hold simultaneously
 - Mutual Exclusion: only one process at a time can use a resource
 - Hold and Wait: process holding one resource is waiting to acquire resource held by another process
 - No Preemption: a resource can be released only be the process holding it after the process completed its task
 - Circular Wait: set of waiting processes such that P_{n-1} is waiting for resource from P_n , and P_n is waiting for P_0
 - “Dining Philosophers” in deadlock

Ch.8 – Main Memory

- Cache sits between main memory and CPU registers
- Base and limit registers define logical address space usable by a process
- Compiled code addresses bind to relocatable addresses
 - Can happen at three different stages
 - Compile time: If memory location known a priori, absolute code can be generated
 - Load time: Must generate relocatable code if memory location not known at compile time
 - Execution time: Binding delayed until run time if the process can be moved during its execution
- Memory-Management Unit (MMU) device that maps virtual to physical address
- Simple scheme uses a relocation register which just adds a base value to address
- Swapping allows total physical memory space of processes to exceed physical memory
 - Def: process swapped out temporarily to backing store then brought back in for continued execution
- Backing store: fast disk large enough to accommodate copies of all memory images
- Roll out, roll in: swapping variant for priority-based scheduling.
 - Lower priority process swapped out so that higher priority process can be loaded
- Solutions to Dynamic Storage-Allocation Problem:
 - First-fit: allocate the first hole that is big enough
 - Best-fit: allocate the smallest hole that is big enough (must search entire list) → smallest leftover hole
 - Worst-fit: allocate the largest hole (search entire list) → largest leftover hole
- External Fragmentation: total memory space exists to satisfy request, but is not contiguous
 - Reduced by compaction: relocate free memory to be together in one block
 - Only possible if relocation is dynamic
- Internal Fragmentation: allocated memory may be slightly larger than requested memory
- Physical memory divided into fixed-sized frames: size is power of 2, between 512 bytes and 16 MB
- Logical memory divided into same sized blocks: pages
- Page table used to translate logical to physical addresses
 - Page number (p): used as an index into a page table
 - Page offset (d): combined with base address to define the physical memory address
- Free-frame list is maintained to keep track of which frames can be allocated

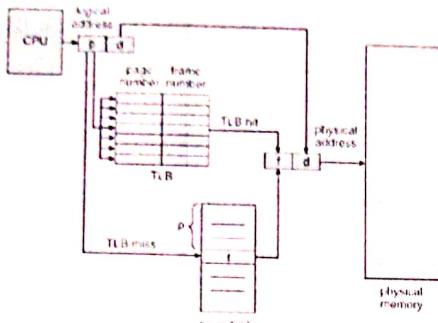


For given logical address space 2^m and page size 2^n

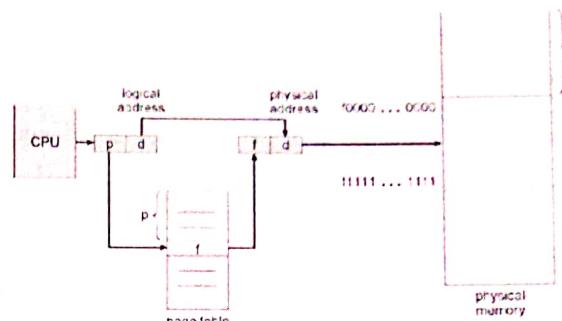


Ch.8 – Main Memory Continued

- Transition Look-aside Buffer (TLB) is a CPU cache that memory management hardware uses to improve virtual address translation speed
 - Typically small – 64 to 1024 entries
 - On TLB miss, value loaded to TLB for faster access next time
 - TLB is associative – searched in parallel

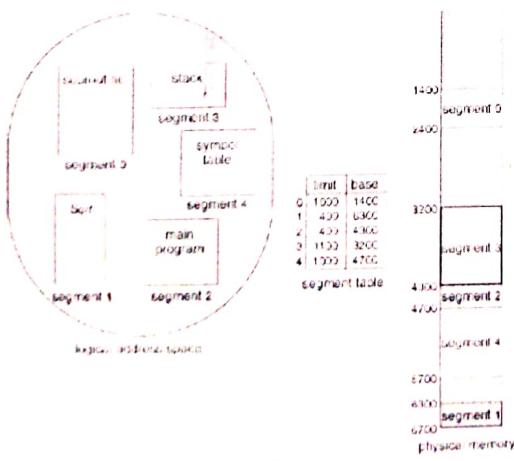


Paging with TLB

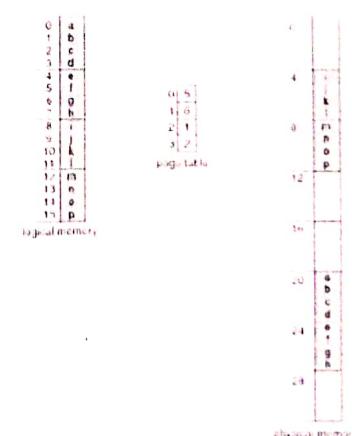


Paging without TLB

- Effective Access Time: $EAT = (1 + \varepsilon) \alpha + (2 + \varepsilon)(1 - \alpha)$
 - ε = time unit, α = hit ratio
- Valid and invalid bits can be used to protect memory
 - “Valid” if the associated page is in the process' logical address space, so it is a legal page
- Can have multilevel page tables (paged page tables)
- Hashed Page Tables: virtual page number hashed into page table
 - Page table has chain of elements hashing to the same location
 - Each element has (1) virtual page number, (2) value of mapped page frame, (3) a pointer to the next element
 - Search through the chain for virtual page number
- Segment table – maps two-dimensional physical addresses
 - Entries protected with valid bits and r/w/x privileges



Segmentation example



Page table example

Ch.9 – Virtual Memory

- Virtual memory: separation of user logical memory and physical memory
 - Only part of program needs to be in memory for execution → logical address space > physical address space
 - Allows address spaces to be shared by multiple processes → less swapping
 - Allows pages to be shared during fork(), speeding process creation
- Page fault results from the first time there is a reference to a specific page → traps the OS
 - Must decide to abort if the reference is invalid, or if the desired page is just not in memory yet
 - If the latter: get empty frame, swap page into frame, reset tables to indicate page now in memory, set validation bit, restart instruction that caused the page fault
 - If an instruction accesses multiple pages near each other → less “pain” because of locality of reference
- Demand Paging only brings a page into memory when it is needed → less I/O and memory needed
 - Lazy swapper – never swaps a page into memory unless page will be needed
 - Could result in a lot of page-faults
 - Performance: $EAT = [(1-p) * \text{memory access} + p * (\text{page fault overhead} + \text{swap page out} + \text{swap page in} + \text{restart overhead})]$; where Page Fault Rate 0 " p " 1
 - if $p = 0$, no page faults; if $p = 1$, every reference is a fault
 - Can optimize demand paging by loading entire process image to swap space at process load time
- Pure Demand Paging: process starts with no pages in memory
- Copy-on-Write (COW) allows both parent and child processes to initially share the same pages in memory
 - If either process modifies a shared page, only then is the page copied
- Modify (dirty) bit can be used to reduce overhead of page transfers → only modified pages written to disk
- When a page is replaced, write to disk if it has been marked dirty and swap in desired page
- Pages can be replaced using different algorithms: FIFO, LRU (below)
 - Stack can be used to record the most recent page references (LRU is a “stack” algorithm)

reference string

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2		2		4	4	4	0		1		1		1		1	
	0	0	0		0		0	0	3	3		3		0		0		0	
	1	1	1		3		3	2	2	2		2		2		2		7	

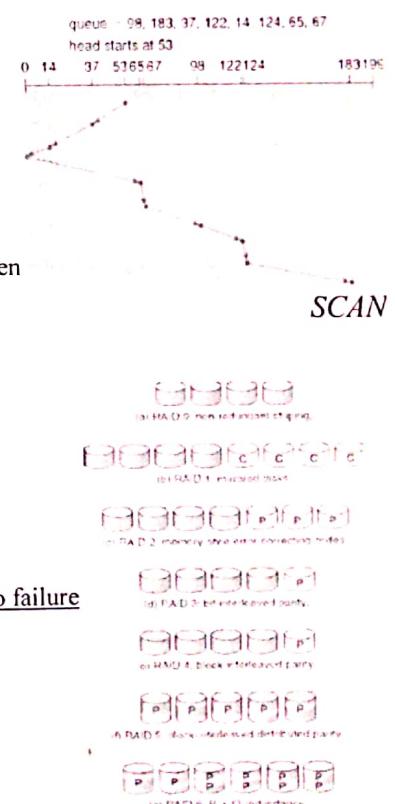
page frames

- Second chance algorithm uses a reference bit
 - If 1, decrement and leave in memory
 - If 0, replace next page
- Fixed page allocation: Proportional allocation – Allocate according to size of process
 - s_i = size of process P_i , $S = \sum s_i$, m = total number of frames, a_i – allocation for P_i
 - $a_i = (s_i/S) * m$
- Global replacement: process selects a replacement frame from set of all frames
 - One process can take frame from another
 - Process execution time can vary greatly
 - Greater throughput
- Local replacement: each process selects from only its own set of allocated frames
 - More consistent performance
 - Possible under-utilization of memory
- Page-fault rate is very high if a process does not have “enough” pages
 - Thrashing: a process is busy swapping pages in and out → minimal work is actually being performed
- Memory-mapped file I/O allows file I/O to be treated as routine memory access by mapping a disk block to a page

- in memory
- I/O Interlock: Pages must sometimes be locked into memory

Ch.10 – Mass-Storage Systems

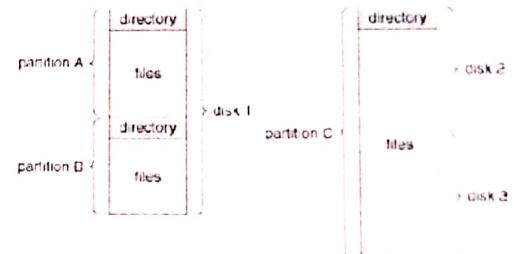
- Magnetic disks provide bulk of secondary storage – rotate at 60 to 250 times per second
 - Transfer rate: rate at which data flows between drive and computer
 - Positioning time (random-access time) is time to move disk arm to desired cylinder (seek time) and time for desired sector to rotate under the disk head (rotational latency)
 - Head crash: disk head making contact with disk surface
- Drive attached to computer's I/O bus – EIDE, ATA, SATA, USB, etc.
 - Host controller uses bus to talk to disk controller
- Access latency = Average access time = average seek time + average latency (fast ~5ms, slow ~14.5ms)
- Average I/O time = avg. access time + (amount to transfer / transfer rate) + controller overhead
 - Ex: to transfer a 4KB block on a 7200 RPM disk with a 5ms average seek time, 1Gb/sec transfer rate with a .1ms controller overhead = $5\text{ms} + 4.17\text{ms} + 4\text{KB} / 1\text{Gb/sec} + 0.1\text{ms} = 9.27\text{ms} + .12\text{ms} = 9.39\text{ms}$
- Disk drives addressed as 1-dimensional arrays of logical blocks
 - 1-dimensional array is mapped into the sectors of the disk sequentially
- Host-attached storage accessed through I/O ports talking to I/O buses
 - Storage area network (SAN): many hosts attach to many storage units, common in large storage environments
 - Storage made available via LUN masking from specific arrays to specific servers
- Network attached storage (NAS): storage made available over a network rather than local connection
- In disk scheduling, want to minimize seek time; Seek time is proportional to seek distance
- Bandwidth is (total number of bytes transferred) / (total time between first request and completion of last transfer)
- Sources of disk I/O requests: OS, system processes, user processes
 - OS maintains queue of requests, per disk or device
- Several algorithms exist to schedule the servicing of disk I/O requests
 - FCFS, SSTF (shortest seek time first), SCAN, CSCAN, LOOK, CLOOK
 - SCAN/elevator: arm starts at one end and moves towards other end servicing requests as it goes, then reverses direction
 - CSCAN: instead of reversing direction, immediately goes back to beginning
 - LOOK/CLOOK: Arm only goes as far as the last request in each directions, then reverses immediately
- Low level/physical formatting: dividing a disk into sectors that the disk controller can read and write – usually 512 bytes of data
- Partition: divide disk into one or more groups of cylinders, each treated as logical disk
- Logical formatting: “making a file system”
- Increase efficiency by grouping blocks into clusters - Disk I/O is performed on blocks
 - Boot block initializes system - bootstrap loader stored in boot block
- Swap-space: virtual memory uses disk space as an extension of main memory
 - Kernel uses swap maps to track swap space use
- RAID: Multiple disk drives provide reliability via redundancy – increases mean time to failure
 - Disk striping uses group of disks as one storage unit
 - Mirroring/shadowing (RAID 1) – keeps duplicate of each disk
 - Striped mirrors (RAID 1+0) or mirrored striped (RAID 0+1) provides high performance/reliability
 - Block interleaved parity (RAID 4, 5, 6) uses much less redundancy
- Solaris ZFS adds checksums of all data and metadata – detect if object is the right one and whether it changed
- Tertiary storage is usually built using removable media – can be WORM or Read-only, handled like fixed disks
- Fixed disk usually more reliable than removable disk or tape drive
- Main memory is much more expensive than disk storage



Ch.11 – File-System Interface

- File – Uniform logical view of information storage (no matter the medium)
 - Mapped onto physical devices (usually nonvolatile)
 - Smallest allotment of nameable storage
 - Types: Data (numeric, character, binary), Program, Free form, Structured
 - Structure decided by OS and/or program/programmer
- Attributes:
 - Name: Only info in human-readable form
 - Identifier: Unique tag, identifies file within the file system
 - Type, Size
 - Location: pointer to file location
 - Time, date, user identification
- File is an abstract data type
- Operations: create, write, read, reposition within file, delete, truncate
- Global table maintained containing process-independent open file information: open-file table
 - Per-process open file table contains pertinent info, plus pointer to entry in global open file table
- Open file locking: mediates access to a file (shared or exclusive)
 - Mandatory – access denied depending on locks held and requested
 - Advisory – process can find status of locks and decide what to do
- File type can indicate internal file structure
- Access Methods: Sequential access, direct access
 - Sequential Access: tape model of a file
 - Direct Access: random access, relative access
- Disk can be subdivided into partitions; disks or partitions can be RAID protected against failure.
 - Can be used raw without a file-system or formatted with a file system
 - Partitions also known as minidisks, slices

file type	usual extension	function
executable	exe, com, bin, or more	ready-to-run machine-language program
object	cobj.o	compiled machine language, not linked
source code	c, cc, java, pub, usen, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data documents
word processor	wp, txt, rtf, doc	various word processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	ara, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information



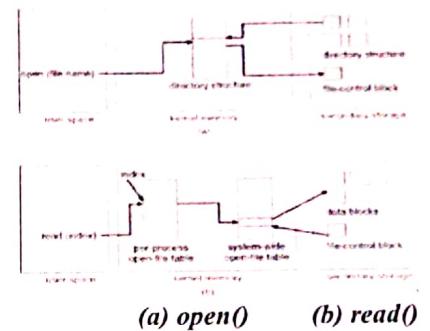
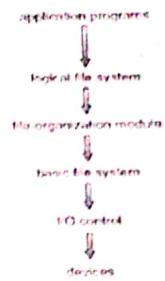
File-System Organization

- Volume contains file system: also tracks file system's info in device directory or volume table of contents
- File system can be general or special-purpose. Some special purpose FS:
 - tmpfs – temporary file system in volatile memory
 - objfs – virtual file system that gives debuggers access to kernel symbols
 - ctfs – virtual file system that maintains info to manage which processes start when system boots
 - lofs – loop back file system allows one file system to be accessed in place of another
 - procfs – virtual file system that presents information on all processes as a file system
- Directory is similar to symbol table – translating file names into their directory entries
 - Should be efficient, convenient to users, logical grouping
 - Tree structured is most popular – allows for grouping
 - Commands for manipulating: remove – rm<file-name>; make new sub directory - mkdir<dir-name>
- Current directory: default location for activities – can also specify a path to perform activities in
- Acyclic-graph directories adds ability to directly share directories between users
 - Acyclic can be guaranteed by: only allowing shared files, not shared sub directories; garbage collection; mechanism to check whether new links are OK
- File system must be mounted before it can be accessed – kernel data structure keeps track of mount points
- In a file sharing system User IDs and Group IDs help identify a user's permissions
- Client-server allows multiple clients to mount remote file systems from servers – NFS (UNIX), CIFS (Windows)
- Consistency semantics specify how multiple users are to access a shared file simultaneously – similar to synchronization algorithms from Ch.7
 - One way of protection is Controlled Access: when file created, determine r/w/x access for users/groups

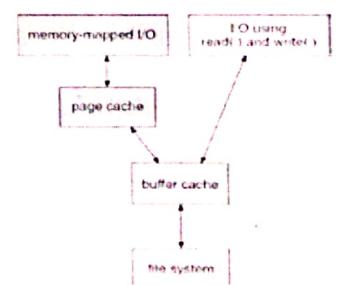
Ch.12 – File System Implementation

- File system resides on secondary storage – disks; file system is organized into layers →
- File control block: storage structure consisting of information about a file (exist per-file)
- Device driver: controls the physical device; manage I/O devices
- File organization module: understands files, logical addresses, and physical blocks
 - Translates logical block number to physical block number
 - Manages free space, disk allocation
- Logical file system: manages metadata information – maintains file control blocks
- Boot control block: contains info needed by system to boot OS from volume
- Volume control block: contains volume details; ex: total # blocks, # free blocks, block size, free block pointers
- Root partition: contains OS; mounted at boot time
- For all partitions, system is consistency checked at mount time
 - Check metadata for correctness – only allow mount to occur if so
- Virtual file systems provide object-oriented way of implementing file systems
- Directories can be implemented as Linear Lists or Hash Tables
 - Linear list of file names with pointer to data blocks – simple but slow
 - Hash table – linear list with hash data structure – decreased search time
 - Good if entries are fixed size
 - Collisions can occur in hash tables when two file names hash to same location
- Contiguous allocation: each file occupies set of contiguous blocks
 - Simple, best performance in most cases; problem – finding space for file, external fragmentation
 - Extent based file systems are modified contiguous allocation schemes – extent is allocated for file allocation
- Linked Allocation: each file is a linked list of blocks – no external fragmentation
 - Locating a block can take many I/Os and disk seeks
- Indexed Allocation: each file has its own index block(s) of pointers to its data blocks
 - Need index table; can be random access; dynamic access without external fragmentation but has overhead
- Best methods: linked good for sequential, not random; contiguous good for sequential and random
- File system maintains free-space list to track available blocks/clusters
- Bit vector or bit map (n blocks): block number calculation → (#bits/word)*(# 0-value words)+(offset for 1st bit)
- Example:

block size = 4KB = 212 bytes
disk size = 240 bytes (1 terabyte)
$n = 240/212 = 228$ bits (or 256 MB)
if clusters of 4 blocks -> 64MB of memory

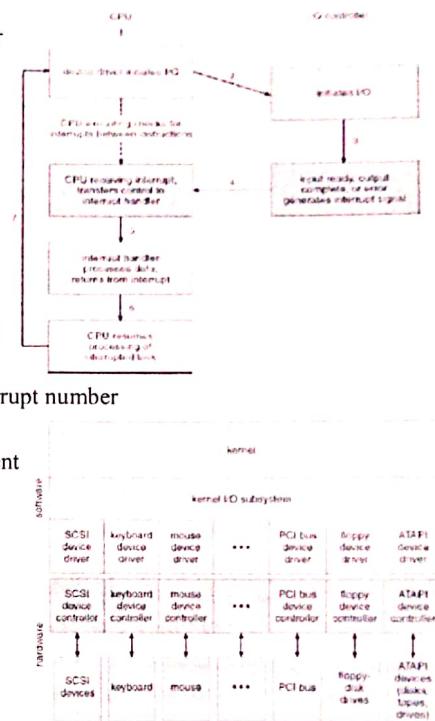


- Space maps (used in ZFS) divide device space into metaslab units and manages metaslabs
 - Each metaslab has associated space map
- Buffer cache – separate section of main memory for frequently used blocks
- Synchronous writes sometimes requested by apps or needed by OS – no buffering
- Asynchronous writes are more common, buffer-able, faster
- Free-behind and read-ahead techniques to optimize sequential access
- Page cache caches pages rather than disk blocks using virtual memory techniques and addresses
 - Memory mapped I/O uses page cache while routine I/O through file system uses buffer (disk) cache
- Unified buffer cache: uses same page cache to cache both memory-mapped pages and ordinary file system I/O to avoid double caching



Ch.13 – I/O Systems

- Device drivers encapsulate device details – present uniform device access interface to I/O subsystem
- Port: connection point for device
- Bus: daisy chain or shared direct access
- Controller (host adapter): electronics that operate port, bus, device – sometimes integrated
 - Contains processor, microcode, private memory, bus controller
- Memory-mapped I/O: device data and command registers mapped to processor address space
 - Especially for large address spaces (graphics)
- Polling for each byte of data – busy-wait for I/O from device
 - Reasonable for fast devices, inefficient for slow ones
 - Can happen in 3 instruction cycles
- CPU interrupt-request line is triggered by I/O devices – interrupt handler receives interrupts
 - Handler is maskable to ignore or delay some interrupts
 - Interrupt vector dispatches interrupt to correct handler – based on priority; some nonmaskable
 - Interrupt chaining occurs if there is more than one device at the same interrupt number
 - Interrupt mechanism is also used for exceptions
- Direct memory access is used to avoid programmed I/O for large data movement
 - Requires DMA controller
 - Bypasses CPU to transfer data directly between I/O device and memory
- Device driver layer hides differences among I/O controllers from kernel
- Devices vary in many dimensions: character stream/block, sequential/random access, synchronous/asynchronous, sharable/dedicated, speed, rw/ro/wo
- Block devices include disk drives: Raw I/O, Direct I/O
 - Commands include read, write, seek
- Character devices include keyboards, mice, serial ports
 - Commands include get(), put()
- Network devices also have their own interface; UNIX and Windows NT/9x/2000 include socket interface
 - Approaches include pipes, FIFOs, streams, queues, mailboxes
- Programmable interval timer: used for timings, periodic interrupts
- Blocking I/O: process suspended until I/O completed – easy to use and understand, not always best method
- Nonblocking I/O: I/O call returns as much as available – implemented via multi-threading, returns quickly
- Asynchronous: process runs while I/O executes – difficult to use, process signaled upon I/O completion
- Spooling: hold output for a device – if device can only serve one request at a time (ex: printer)
- Device Reservation: provides exclusive access to a device – must be careful of deadlock
- Kernel keeps state info for I/O components, including open file tables, network connections, character device states
 - Complex data structures track buffers, memory allocation, “dirty” blocks
- STREAM: full-duplex communication channel between user-level process and device in **UNIX**
 - Each module contains read queue and write queue
 - Message passing used to communicate between queues – Flow control option to indicate available or busy
 - Asynchronous internally, synchronous where user process communicates with stream head
- I/O is a major factor in system performance – demand on CPU, context switching, data copying, network traffic



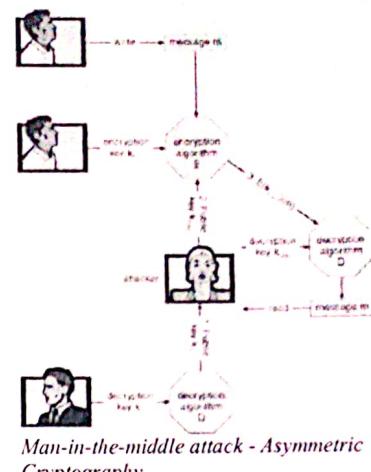
Ch.14 – Protection

- Principle of least privilege: programs, users, systems should be given just enough privileges to perform their tasks
- Access-right = <obj-name, rights-set> w/ rights-set is subset of all valid operations performable on the object
 - Domain: set of access-rights
 - UNIX system consists of 2 domains: user, supervisor
 - MULTICS domain implementation (domain rings) – if $j < i \rightarrow D_i \sqsupseteq D_j$
- Access matrix: rows represent domains, columns represent objects
 - Access(i,j) is the set of operations that a process executing in Domain_i can invoke on Object_j
 - Can be expanded to dynamic protection
- Access matrix design separates mechanism from policy
 - Mechanism: OS provides access-matrix and rules – ensures matrix is only manipulated by authorized users
 - Policy: User dictates policy – who can access what object and in what mode
- Solaris 10 uses role-based access control (RBAC) to implement least privilege
- Revocation of access rights
 - Access list: delete access rights from access list – simple, immediate
 - Capability list: required to locate capability in system before capability can be revoked – reacquisition, back-pointers, indirection, keys
- Language-Based Protection: allows high-level description of policies for the allocation and use of resources
 - Can provide software for protection enforcement when hardware-supported checking is unavailable

object domain	F_1	F_2	F_3	printer
D_1	read			read
D_2				print
D_3		read	execute	
D_4	read write			read write

Ch.15 – Security

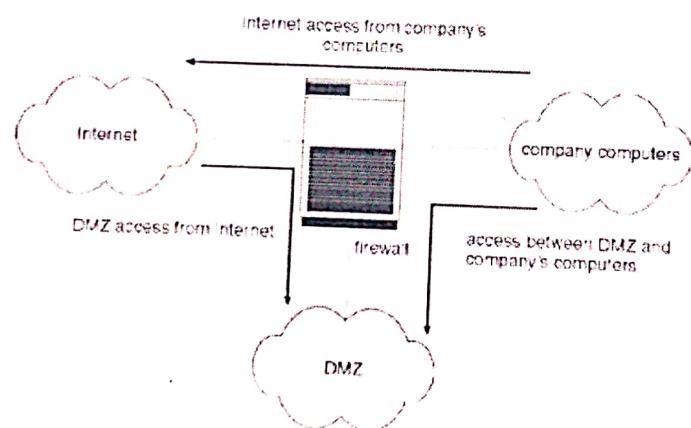
- System secure when resources used and accessed as intended under all circumstances
- Attacks can be accidental or malicious
 - Easier to protect against accidental than malicious misuse
- Security violation categories:
 - Breach of confidentiality – unauthorized reading of data
 - Breach of integrity – unauthorized modification of data
 - Breach of availability – unauthorized destruction of data
 - Theft of service – unauthorized use of resources
 - Denial of service – prevention of legitimate use
- Methods of violation:
 - Masquerading – pretending to be an authorized user
 - Man-in-the-middle – intruder sits in data flow, masquerading as sender to receiver and vice versa
 - Session hijacking – intercept and already established session to bypass authentication
- Effective security must occur at four levels: physical, human, operating system, network
- Program threats: trojan horse (spyware, pop-up, etc.), trap door, logic bomb, stack and buffer overflow
- Viruses: code fragment embedded in legitimate program; self-replicating
 - Specific to CPU architecture, OS, applications
 - Virus dropper: inserts virus onto the system
- Windows is the target for most attacks – most common, everyone is administrator
- Worms: use spawn mechanism – standalone program
- Port scanning: automated attempt to connect to a range of ports on one or a range of IP addresses
 - Frequently launched from zombie systems to decrease traceability
- Denial of service: overload targeted computer preventing it from doing useful work
- Cryptography: means to constrain potential senders and/or receivers – based on keys
 - Allows for confirmation of source, receipt by specified destination, trust relationship
- Encryption: [K of keys], [M of messages], [C of ciphertexts], function E:K to encrypt, function D:K to decrypt
 - Can have symmetric and asymmetric (distributes public encryption key, holds private decipher key) encryption
 - Asymmetric is much more compute intensive – not used for bulk data transaction
 - Keys can be stored on a key ring
- Authentication: constraining a set of potential senders of a message
 - Helps to prove that the message is unmodified
 - Hash functions are basis of authentication
 - Creates small, fixed-size block of data (message digest, hash value)
- Symmetric encryption used in message-authentication code (MAC)
- Authenticators produced from authentication algorithm are digital signatures
- Authentication requires fewer computations than encryption methods
- Digital Certificates: proof of who or what owns a public key
- Defense in depth: most common security theory – multiple layers of security
- Can attempt to detect intrusion:
 - Signature-based: detect “bad patterns”
 - Anomaly detection: spots differences from normal behavior
 - Both can report false positives or false negatives
 - Auditing, accounting, and logging specific system or network activity



Man-in-the-middle attack - Asymmetric Cryptography

Ch.15 – Security Continued

- Firewall: placed between trusted and untrusted hosts
 - Limits network access between the two domains
 - Can be tunneled or spoofed
- Personal firewall is software layer on given host
 - Can monitor/limit traffic to/from host
- Application proxy firewall: Understands application protocol and can control them
- System-call firewall: Monitors all important system calls and apply rules and restrictions to them



Part 1: Overview

1. Introduction

1 Care sunt cele trei scopuri principale ale unui sistem de operare?

Răspuns:

Cele trei cauze principale sunt:

- Pentru a oferi un mediu pentru un utilizator de computer pentru a executa programe pe hardware-ul computerului în mod convenabil și eficient.
 - Pentru a aloca resursele separate ale calculatorului, după cum este necesar, pentru a rezolva problema dată. Procesul de alocare ar trebui să fie cât mai echitabil și mai eficient posibil.
 - Ca program de control servește două funcții majore: (1) supravegherea execuției programelor de utilizator pentru a preveni erorile și utilizarea necorespunzătoare a calculatorului și (2) gestionarea operării și controlului dispozitivelor I / O.
- 2 Am subliniat necesitatea ca un sistem de operare să utilizeze în mod eficient hardware-ul de calcul. Când este oportun ca sistemul de operare să renunțe la acest principiu și să "piardă" resursele? De ce un astfel de sistem nu este deloc risipitor?

Răspuns:

Sistemele cu un singur utilizator ar trebui să maximizeze utilizarea sistemului pentru utilizator. O interfață grafică poate să "piardă" ciclurile procesorului, dar optimizează interacțiunea utilizatorului cu sistemul.

3 Care este principala dificultate pe care un programator trebuie să o depășească scriind un sistem de operare pentru un mediu în timp real?

Răspuns:

Principala dificultate este păstrarea sistemului de operare în limitele de timp fixe ale unui sistem în timp real. Dacă sistemul nu realizează o sarcină într-un anumit interval de timp, poate provoca o defecțiune a întregului sistem pe care îl execută. Prin urmare, atunci când scriem un sistem de operare pentru un sistem în timp real, scriitorul trebuie să fie sigur că schemele sale de planificare nu permit timpul de răspuns să depășească constrângerea de timp.

4 Înțând cont de diferitele definiții ale sistemului de operare, luați în considerare dacă sistemul de operare trebuie să includă aplicații precum browsere Web și programe de poștă electronică. Argumentează atât că ar trebui și că nu ar trebui, și să vă sprijine răspunsurile.

Răspuns:

Un argument în favoarea incluziei aplicațiilor populare în sistemul de operare este acela că, dacă aplicația este încorporată în sistemul de operare, este probabil să fie mai bine să profite de caracteristicile din kernel și, prin urmare, să aibă avantaje de performanță față de o aplicație care rulează în afara din kernel. Argumentele împotriva aplicațiilor de încorporare din sistemul de operare, de obicei, domină totuși: (1) aplicațiile sunt aplicații - și nu fac parte dintr-un sistem de operare, (2) orice avantaje de performanță ale funcționării în kernel sunt compensate de vulnerabilitățile de securitate; la un sistem de operare umflat.

5 Cum face distincția dintre modul kernel și modul de utilizator ca o formă rudimentară de sistem de protecție (securitate)?

Răspuns:

Distinctia dintre modul kernel și modul utilizator oferă o formă rudimentară de protecție în modul următor. Anumite instrucțiuni ar putea fi executate numai atunci când CPU-ul este în modul kernel. În mod similar, dispozitivele hardware ar putea fi accesate numai atunci când programul se execută în modul kernel. Controlul asupra momentului în care intreruperile ar putea fi activate sau dezactivate este posibil și numai atunci când CPU-ul este în modul kernel. În consecință, CPU-ul are o capacitate foarte limitată atunci când se execută în modul de utilizare, impunând astfel protecția resurselor critice.

6 Care dintre următoarele instrucțiuni ar trebui să fie privilegiat?

- a) Setați valoarea cronometrului.
- b) Citiți ceasul.
- c) Eliminați memoria.
- d) Eliberați o instrucțiune pentru capcană.
- e) Dezactivați intreruperile.
- f) Modificați intrările în tabelul de stare a dispozitivului.
- g) Treceți de la modul utilizator la modul kernel.
- h) Accesați dispozitivul I / O.

Răspuns:

Următoarele operații trebuie să fie privilegiate: setați valoarea temporizatorului, ștergeți memoria, dezactivați intreruperile, modificați intrările din tabela de stare a dispozitivului, accesați dispozitivul I / O. Restul poate fi efectuat în modul utilizator.

7 Unele computere timpurii au protejat sistemul de operare prin plasarea acestuia într-o partitură de memorie care nu a putut fi modificată nici de lucrarea utilizatorului, nici de sistemul de operare în sine. Descrieți două dificultăți pe care le credeți că ar putea apărea cu o astfel de schemă.

Răspuns:

Datele solicitate de sistemul de operare (parole, controale de acces, informații contabile și aşa mai departe) ar trebui să fie stocate sau transmise prin memorie neprotejată și astfel să fie accesibile utilizatorilor neautorizați.

8 Unele procesoare oferă mai mult de două moduri de funcționare. Care sunt două posibile utilizări ale acestor multiple moduri?

Răspuns:

Deși majoritatea sistemelor disting între modurile de utilizare și kernel, unele CPU-uri au suportat mai multe moduri. Modele multiple ar putea fi utilizate pentru a oferi o politică de securitate mai fină. De exemplu, mai degrabă decât să faceți distincția între modul de utilizare și modul kernel, ați putea distinge între diferitele tipuri de mod de utilizator. Poate că utilizatorii aparținând aceluiași grup ar putea executa codul celuilalt. Mașina ar intra într-un mod specific când unul dintre acești utilizatori rula cod. Când aparatul era în acest mod, un membru al grupului ar putea rula codul aparținând oricui altcuiva din grup. O altă posibilitate ar fi să oferim distincții diferite în cadrul codului kernelului. De exemplu, un mod specific ar putea permite driverelor dispozitivelor USB să ruleze. Acest lucru ar însemna că dispozitivele USB ar putea fi întreținute fără a trebui să treacă la modul kernel, permitând astfel în mod esențial ca driverele dispozitivelor USB să ruleze într-un mod cvasi-utilizator / kernel.

9 Timerele pot fi utilizate pentru a calcula ora curentă. Furnizați o scurtă descriere a modului în care s-ar putea realiza acest lucru.

Răspuns:

Un program ar putea utiliza următoarea abordare pentru a calcula ora curentă utilizând întreruperile temporizatorului. Programul ar putea seta un temporizator pentru ceva timp în viitor și să meargă la culcare. Când este trezit de întrerupere, ar putea actualiza starea sa locală, pe care o utilizează pentru a urmări numărul de întreruperi pe care le-a primit până acum. Ar putea apoi să repetă acest proces de a seta continuu întreruperile temporizatorului și actualizarea stării locale atunci când întreruperile sunt efectiv ridicate.

10 Dați două motive pentru care cache-urile sunt utile. Ce probleme rezolvă? Ce probleme provoacă? Dacă o memorie cache poate fi făcută la fel de mare ca și dispozitivul pentru care este cache (de exemplu, un cache la fel de mare ca un disc), de ce să nu o faceți atât de mare și să eliminați dispozitivul?

Răspuns:

Cache-urile sunt utile atunci când două sau mai multe componente necesită schimb de date, iar componentele efectuează transferuri la viteze diferite. Cache-urile rezolvă problema transferului prin furnizarea unui buffer de viteză intermediară între componente. Dacă dispozitivul rapid află datele de care are nevoie în memoria cache, nu este nevoie să așteptați dispozitivul mai lent. Datele din memoria cache trebuie păstrate în conformitate cu datele din componentă. Dacă o componentă are o modificare de valoare a datelor, iar originea este, de asemenea, în memoria cache, cache-ul trebuie, de asemenea, să fie actualizat. Aceasta este o problemă deosebită pentru sistemele multiprocesor în care mai multe procese pot accesa un element. O componentă poate fi eliminată printr-o cache de dimensiuni egale, dar numai dacă: (a) cache-ul și componenta au o capacitate echivalentă de salvare a statului (adică dacă componenta își păstrează datele când electricitatea este îndepărtată, cache-ul trebuie să păstreze datele) și (b) memoria cache este accesibilă, deoarece stocarea mai rapidă tinde să fie mai costisitoare.

11 Distingeți între modelele client-server și peer-to-peer ale sistemelor distribuite.

Răspuns:

Modelul client-server distinge cu fermitate rolurile clientului și serverului. În cadrul acestui model, clientul solicită servicii furnizate de server. Modelul peer-to-peer nu are roluri atât de stricte. De fapt, toate nodurile din sistem sunt considerate colegi și pot acționa astfel fie ca clienți, fie ca servere - sau ambele. Un nod poate solicita un serviciu de la un alt coleg, sau nodul poate furniza de fapt un astfel de serviciu celorlalți colegi din sistem. De exemplu, să luăm în considerare un sistem de noduri care împărtășesc rețete de gătit. În cadrul modelului client-server, toate rețetele sunt stocate împreună cu serverul. Dacă un client dorește să acceseze o rețetă, trebuie să solicite rețeta de la serverul specificat. Folosind modelul peer-to-peer, un nod de tip peer poate solicita altă noduri de tip peer pentru rețeta specificată. Nodul (sau, probabil, nodurile) cu rețeta cerută îl poate furniza nodului solicitant. Observați cum fiecare coleg poate acționa atât ca un client (poate cere rețete), cât și ca un server (poate oferi rețete).

2. Operating-System Structures

1. Care este scopul apelurilor de sistem?

Răspuns:

Apelurile de sistem permit proceselor la nivel de utilizator să solicite servicii ale sistemului de operare.

2. Care sunt cele cinci activități majore ale unui sistem de operare în ceea ce privește managementul proceselor?

Răspuns:

Cele cinci activități majore sunt:

- a) Crearea și ștergerea proceselor de utilizator și de sistem
- b) Suspendarea și reluarea proceselor
- c) Furnizarea de mecanisme pentru sincronizarea proceselor
- d) Furnizarea de mecanisme pentru comunicarea proceselor
- e) Furnizarea de mecanisme de manipulare a blocajelor

3. Care sunt cele trei activități majore ale unui sistem de operare în ceea ce privește gestionarea memoriei?

Răspuns:

Cele trei activități majore sunt:

- a) Urmăriți ce părți din memorie sunt utilizate în prezent și de cine.
- b) Decideți care procese vor fi încărcate în memorie atunci când spațiul de memorie devine disponibil.
- c) Alocați și dezaxați spațiul de memorie după cum este necesar.

4. Care sunt cele trei activități majore ale unui sistem de operare în ceea ce privește gestionarea stocării secundare?

Răspuns:

Cele trei activități majore sunt:

- Gestiona spațiului liber.
- Alocare depozitare.
- Programarea discurilor.

5. Care este scopul interpretului de comandă? De ce este de obicei separat de kernel?

Răspuns:

Citește comenzi de la utilizator sau dintr-un fișier de comenzi și le execută, de obicei transformându-le în unul sau mai multe apele de sistem. De obicei, nu face parte din kernel deoarece interpretul de comandă este supus modificărilor.

6. Ce apele de sistem trebuie să fie executate de un interpret de comandă sau shell pentru a începe un nou proces?

Răspuns:

În sistemele Unix, trebuie efectuat un apel sistem de furcă, urmat de un apel sistem executiv, pentru a începe un nou proces. Apelul de furcă clonează procesul de execuție curent, în timp ce apelul executiv suprapune un nou proces bazat pe un executabil diferit în timpul procesului de apelare.

7. Care este scopul programelor de sistem?

Răspuns:

programele de sistem pot fi considerate ca pachete de apeluri utile de sistem. Acestea oferă utilizatorilor funcționalitatea de bază, astfel încât utilizatorii să nu aibă nevoie să scrie propriile programe pentru a rezolva probleme comune.

8. Care este principalul avantaj al abordării stratificate a designului de sistem? Care sunt dezavantajele utilizării abordării stratificate?

Răspuns:

Ca în toate cazurile de proiectare modulară, proiectarea unui sistem de operare într-un mod modular are mai multe avantaje. Sistemul este mai ușor de depanat și modificat deoarece modificările afectează numai secțiuni limitate ale sistemului, în loc să atingă toate secțiunile sistemului de operare. Informațiile sunt păstrate numai acolo unde sunt necesare și sunt accesibile numai într-o zonă definită și restricționată, astfel încât orice erori care afectează aceste date trebuie să fie limitate la un anumit modul sau strat.

9. Listează cinci servicii furnizate de un sistem de operare și explică modul în care fiecare creează comoditate pentru utilizatori. În ce cazuri ar fi imposibil ca programele la nivel de utilizator să furnizeze aceste servicii? Explică-ți răspunsul.

Răspuns:

Cele cinci servicii sunt:

- Execuția programului. Sistemul de operare încarcă conținutul (sau secțiunile) unui fișier în memorie și începe executarea acestuia. Un program la nivel de utilizator nu putea avea încredere în alocarea corectă a timpului CPU.
- Operațiuni I / O. Discurile, casetele, liniile seriale și alte dispozitive trebuie comunicate cu un nivel foarte scăzut. Utilizatorul trebuie să specifice doar dispozitivul și operația de efectuat pe acesta, în timp ce sistemul convertește cererea în comenzi specifice dispozitivului sau controlerului. La programele la nivel de utilizator nu se poate avea încredere să acceseze numai dispozitivele la care ar trebui să aibă acces și să le acceseze numai atunci când acestea nu sunt folosite în alt mod.
- File manipularea sistemului. Există multe detalii în crearea, ștergerea, alocarea și denumirea fișierelor pe care utilizatorii nu ar trebui să le efectueze. Blocurile de spațiu pe disc sunt utilizate de fișiere și trebuie urmărite. Ștergerea unui fișier necesită eliminarea informațiilor despre fișierul de nume și eliberarea blocurilor alocate. De asemenea, trebuie verificate și protecțiile pentru a asigura accesul corect la fișiere. Programele de utilizatori nu au putut asigura respectarea metodelor de protecție și nu au încredere în alocarea blocurilor libere și blocarea blocării în timpul ștergerii fișierelor.
- Comunicații. Transmiterea mesajelor între sisteme necesită ca mesajele să fie transformate în pachete de informații, trimise la controlerul de rețea, transmise pe un mediu de comunicații și reasamblate de sistemul de destinație. Ordonarea pachetelor și corectarea datelor trebuie să aibă loc. Din nou, programele de utilizatori ar putea să nu coordoneze accesul la dispozitivul de rețea sau ar putea primi pachete destinate altor procese.
- Eroare detectată. Detectarea erorilor apare atât la nivel hardware, cât și la nivel de software. La nivel hardware, toate transferurile de date trebuie inspectate pentru a se asigura că datele nu au fost corupte în timpul transportului. Toate datele despre mass-media trebuie să fie verificate pentru a fi siguri că nu s-au schimbat de când au fost scrise pe suport media. La

nivel de software, media trebuie să fie verificată pentru consistență datelor; de exemplu, dacă numărul de blocuri de stocare alocate și nealocate corespunde numărului total al dispozitivului. Acolo, erorile sunt adesea procese independente (de exemplu, corupția datelor pe un disc), deci trebuie să existe un program global (sistemul de operare) care să gestioneze toate tipurile de erori. De asemenea, prin procesarea erorilor de către sistemul de operare, procesele nu trebuie să conțină coduri pentru a captura și corecta toate erorile posibile ale unui sistem.

10. De ce unele sisteme stochează sistemul de operare în firmware, în timp ce altele stochează pe disc?

Răspuns:

Pentru anumite dispozitive, cum ar fi PDA-urile portabile și telefoanele celulare, este posibil ca un disc cu un sistem de fișiere să nu fie disponibil pentru dispozitiv. În această situație, sistemul de operare trebuie să fie stocat în firmware.

11. Cum ar putea fi proiectat un sistem care să permită alegerea sistemelor de operare de la care să pornească? Ce ar trebui să facă programul de bootstrap?

Răspuns:

Luăți în considerare un sistem care ar dori să ruleze atât Windows XP cât și trei distribuții diferite de Linux (de ex., RedHat, Debian și Mandrake). Fiecare sistem de operare va fi stocat pe disc. În timpul sistemului de boot-up, un program special (pe care îl vom numi managerul de boot) va determina sistemul de operare la care să se inițieze. Aceasta înseamnă că, mai degrabă inițial bootarea la un sistem de operare, managerul de boot va rula mai întâi în timpul pornirii sistemului. Acesta este managerul de boot care este responsabil pentru a determina care sistem să boot. În general, administratorii de booturi trebuie să fie stocați în anumite locații ale hard disk-ului care trebuie recunoscute în timpul pornirii sistemului. Managerii de boot oferă adesea utilizatorului o selecție de sisteme în care să se lanseze; managerii de boot sunt, de asemenea, proiectați în mod obișnuit să ruleze într-un sistem de operare implicit, dacă nu este selectată nicio alegere de către utilizator.

Part 2: Process Management

3. Processes

1. Folosind programul prezentat în Figura 3.30, explicați ce va fi ieșirea la Linia A.

Răspuns:

Rezultatul este încă 5, deoarece copilul își actualizează copia de valoare. Când controlul revine la părinte, valoarea acestuia rămâne la 5.

2. Inclusiv procesul inițial părinte, câte procese sunt create de programul prezentat în Figura 3.31?

Răspuns:

Sunt create 8 procese.

3. Versiunile originale ale sistemului de operare iOS de la Apple nu oferă nici un mijloc de procesare simultană. Discutați despre trei complicații majore pe care procesarea concurențială le adaugă unui sistem de operare.

Răspuns:

Completați

4. Procesorul Sun UltraSPARC are mai multe seturi de registru. Descrieți ce se întâmplă când apare un comutator de context dacă noul context este deja încărcat într-unul din seturile de registru. Ce se întâmplă dacă noul context este în memorie, mai degrabă decât într-un set de registru și toate seturile de registru sunt utilizate?

Răspuns:

Pointerul setat de registrul curent al CPU este modificat pentru a indica setul care conține noul context, care durează foarte puțin timp. Dacă contextul este în memorie, trebuie selectat unul dintre contextele dintr-un set de registru și să fie mutat în memorie, iar noul context trebuie încărcat din memorie în set. Acest proces necesită puțin mai mult timp decât sistemele cu un set de registre, în funcție de modul în care este selectată o victimă înlocuitoare.

5. Când un proces creează un proces nou utilizând operația furcă (), care din următoarea stare este partajată între procesul părinte și procesul copilului?
- a) Stivă
 - b) Heap
 - c) Segmente de memorie partajată

Răspuns:

Numai segmentele de memorie partajată sunt partajate între procesul părinte și procesul copilului nou creionat. Copii ale stiva și halda sunt făcute pentru procesul nou creat.

6. În ceea ce privește mecanismul RPC, luați în considerare semnul "exact o dată" semantic. Algoritmul pentru implementarea acestui semantic se execută corect chiar dacă mesajul ACK înapoi către client este pierdut din cauza unei probleme de rețea? Descrieți succesiunea mesajelor și discutați dacă se păstrează "exact o dată".

Răspuns:

Semantica "exact o dată" se asigură că o procedură de remodelare va fi executată exact o singură dată. Algoritmul general pentru a asigura acest lucru combină o schemă de confirmare (ACK) combinată cu timbre (sau un alt numărător incremental care permite serverului să facă distincția între mesajele dupicate). Strategia generală este ca clientul să trimită serverul RPC pe server împreună cu un marcat de timp. De asemenea, clientul va începe un ceas de expirare. Apoi, clientul va aștepta una dintre cele două apariții: (1) va primi un ACK de la server indicând faptul că procedura de la distanță a fost efectuată sau (2) se va opri. Dacă clientul renunță la timp, acesta presupune că serverul nu a putut efectua procedura de la distanță, astfel încât clientul să invocă RPC a doua oară, trimițând o marcă de timp mai târzie. Clientul nu poate primi ACK pentru unul din două motive: (1) RPC original nu a fost niciodată primit de server sau (2) RPC a fost corect primit și executat de server, dar ACK-ul a fost pierdut. În situația (1), utilizarea ACK-urilor permite serverului să primească și să execute RPC în cele din urmă. În situația (2), serverul va primi un RPC dupicat și va folosi amprenta de timp pentru a-l identifica ca dupicat pentru a nu efectua RPC a doua oară. Este important să rețineți că serverul trebuie să trimită un al doilea ACK înapoi la client pentru a informa clientul că RPC-ul a fost efectuat.

7. Să presupunem că un sistem distribuit este susceptibil la eşecul serverului. Ce mecanisme ar fi necesare pentru a garanta o semantică "exact o dată" pentru executarea RPC?

Răspuns:

Serverul ar trebui să țină evidență în timpul stocării stabile (cum ar fi un jurnal de disc) cu privire la operațiile RPC primite, dacă au fost efectuate cu succes și rezultatele asociate cu operațiunile. Atunci când are loc un accident de server și este recepționat un mesaj RPC, serverul poate verifica dacă RPC-ul a fost efectuat anterior și, prin urmare, garantează semantica "exact o dată" pentru executarea RPC-urilor.

4. Threads

1. Furnizați trei exemple de programare în care multithreading oferă o performanță mai bună decât o soluție cu un singur fișier.
 - a) Un server Web care oferă fiecare cerere într-un fir separat.
 - b) aplicație paralelizată, cum ar fi multiplicarea matricei, în care pot fi prelucrate în paralel diferite părți ale matricei.
 - c) Un program interactiv GUI, cum ar fi un program de depanare în cazul în care un fir este utilizat pentru a monitoriza intrarea utilizatorilor, un alt fir reprezintă aplicația care rulează, iar un al treilea fir monitorizează performanța.
2. Care sunt două diferențe între firele la nivel de utilizator și firele la nivel de kernel? În ce condiții este un tip mai bun decât celălalt?

Răspuns:

- a) Firele la nivel de utilizator nu sunt cunoscute de kernel, în timp ce kernel-ul este conștient de firele kernel-ului.
 - b) Pe sistemele care utilizează maparea M: 1 sau M: N, firele utilizatorilor sunt programate de biblioteca de fire și nucleul programează firele kernel-ului.
 - c) Firele de kernel nu trebuie să fie asociate cu un proces, în timp ce fiecare fir de utilizator aparține unui proces. Firele de kernel sunt, în general, mai scumpe decât cele de utilizator, deoarece acestea trebuie reprezentate cu o structură de date a kernel-ului.
3. Descrieți acțiunile întreprinse de un kernel pentru a schimba contextul între firele kernellevel.

Răspuns:

Schimbarea de context între firele de kernel necesită în mod tipic salvarea valorii registrelor CPU din firul care este deconectat și restaurarea registrelor procesorului noului thread programat.

4. Ce resurse sunt utilizate atunci când este creat un fir? Cum diferă acestea de cele utilizate atunci când este creat un proces?

Răspuns:

Deoarece un fir este mai mic decât un proces, crearea de fire utilizează de obicei mai puține resurse decât crearea de procese. Crearea unui proces necesită alocarea unui bloc de control al procesului (PCB), o structură de date destul de mare. PCB include o hartă de memorie, o listă de fișiere deschise și variabile de mediu. Alocarea și gestionarea hărții de memorie este de obicei cea mai consumatoare de timp. Crearea unui fir sau a unui fir de nucleu implică alocarea unei structuri de date mici pentru a deține un set de registru, o stivă și o prioritate.

5. Să presupunem că un sistem de operare atașează firele la nivel de utilizator la kernel folosind modelul multi-to-many și că procesul de mapare este realizat prin LWP-uri. În plus, sistemul

permite dezvoltatorilor să creeze fir în timp real pentru utilizarea în sistemele în timp real. Este necesar să legați firul în timp real într-un LWP? Explica.

Răspuns:

Da. Sincronizarea este crucială pentru aplicațiile în timp real. Dacă un fir este marcat ca fiind în timp real, dar nu este legat de un LWP, este posibil ca firul să aştepte să fie atașat la un LWP înainte de a fi rulat. Luați în considerare dacă un fir în timp real rulează (este atașat la un LWP) și apoi continuă să blocheze (adică trebuie să efectueze I / O, a fost preempted de un fir de prioritate cu prioritate în timp real, aşteaptă o blocare a excluderii reciproce, etc) în timp ce firul în timp real este blocat, LWP la care a fost atașat a fost atribuit unui alt fir. Când firul în timp real a fost programat să ruleze din nou, trebuie mai întâi să aştepta să fie atașat la un LWP. Prin legarea unui LWP la un thread în timp real, asigurați-vă că firul va putea rula cu întârziere minimă odată ce este programat.

5. Process Synchronization

- În secțiunea 5.4 am menționat că dezactivarea întreruperilor frecvent poate afecta ceasul sistemului. Explicați de ce acest lucru poate apărea și cum pot fi minimezate astfel de efecte.

Răspuns:

Ceasul sistemului este actualizat la fiecare întrerupere a ceasului. Dacă întreruperile au fost dezactivate - în special pentru o perioadă lungă de timp - este posibil ca ceasul sistemului să poată pierde cu ușurință timpul corect. Ceasul de sistem este, de asemenea, utilizat în scopuri de planificare. De exemplu, quantumul timpului pentru un proces este exprimat ca un număr de căpușe de ceas. La fiecare întrerupere a ceasului, planificatorul determină dacă s-a expirat quantumul timpului pentru procesul care rulează în prezent. Dacă întreruperile de ceas au fost dezactivate, planificatorul nu a putut atribui cu exactitate quantumurile de timp. Acest efect poate fi minimezat prin dezactivarea întreruperilor de ceas pentru perioade foarte scurte.

- Explicați de ce Windows, Linux și Solaris implementează mai multe mecanisme de blocare. Descrieți circumstanțele în care folosesc spinlock-uri, încuietori de mutex, semafoare, blocări adaptive de mutex și variabile de stare. În fiecare caz, explicați de ce este necesar mecanismul.

Răspuns:

Aceste sisteme de operare oferă mecanisme diferite de blocare în funcție de nevoile dezvoltatorilor de aplicații. Spinlock-urile sunt utile pentru sistemele multiprocesoare unde un fir poate rula într-o buclă ocupată (pentru o perioadă scurtă de timp), mai degrabă decât să suporte cheltuielile de a fi plasate într-o coadă de somn. Mutexurile sunt utile pentru blocarea resurselor. Solaris 2 folosește mutexuri adaptive, ceea ce înseamnă că mutexul este implementat cu un dispozitiv de blocare pe mașini multiprocesoare. Semaphorele și variabilele de stare sunt instrumente mai potrivite pentru sincronizare atunci când o resursă trebuie menținută pentru o perioadă lungă de timp, deoarece rotația este neficientă pe o durată lungă.

- Care este sensul termenului ocupat de aşteptare? Ce alte tipuri de aşteptare există într-un sistem de operare? Poate fi ocupat de aşteptare să fie evitat cu totul?

Răspuns:

Ocuparea în aşteptare înseamnă că un proces aşteaptă ca o condiție să fie îndeplinită într-o buclă strânsă fără a renunța la procesor. În mod alternativ, un proces ar putea aştepta să renunțe la procesor și să blocheze o afecțiune și să aştepte să fie trezit la un moment potrivit în viitor. Ocuparea

așteptată poate fi evitată, dar implică cheltuielile aferente punerii unui proces în starea de somn și necesitatea de a se trezi atunci când se ajunge la starea corespunzătoare a programului.

4. Explicați de ce spinlock-urile nu sunt potrivite pentru sistemele cu un singur procesor și sunt adesea folosite în sistemele multiprocesor.

Răspuns:

Spinlock-urile nu sunt potrivite pentru sistemele cu un singur procesor, deoarece condiția care va rupe un proces din spinlock poate fi obținută numai prin executarea unui alt proces. Dacă procesul nu renunță la procesor, alte procese nu primesc posibilitatea de a seta starea programului necesară pentru primul proces pentru a face progrese. Într-un sistem multiprocesor, alte procese se execută pe alte procesoare și modifică astfel starea programului pentru a elibera primul proces din spinlock.

5. Arătați că, în cazul în care operațiile semafor () și semnal () nu sunt executate în mod atomic, atunci excluderea reciprocă poate fi încălcată.

Răspuns:

O operație de așteptare scade atomic valoarea asociată unui semafor. Dacă două operațiuni de așteptare sunt executate pe un semafor când valoarea sa este 1, dacă cele două operații nu sunt efectuate în mod atomic, atunci este posibil ca ambele operații să continue să scadă valoarea semaforului, încălcând astfel excluderea reciprocă.

6. Ilustrați modul în care semaforul binar poate fi folosit pentru a implementa excluderea reciprocă între procesele n.

Răspuns:

Procesul n împarte un semaphore, mutex, inițializat la 1. Fiecare proces Pi este organizat după cum urmează:

```
do {  
    wait(mutex);  
    /* critical section */  
    signal(mutex);  
    /* remainder section */  
} while (true);
```

7. Listați trei exemple de blocări care nu sunt legate de un mediu de computere.

Răspuns:

- Două mașini care traversează o punte cu o singură bandă din direcții opuse
- persoană care coboară pe o scară în timp ce o altă persoană se urcă pe scară.
- Două trenuri care se deplasează unul pe celălalt pe aceeași pistă.

8. Este posibil să aveți un impas care implică doar un singur proces? Explică-ți răspunsul.

Răspuns:

Nu. Acest lucru rezultă direct din starea de așteptare și așteptare.

6. CPU Scheduling

1. Un algoritm de programare CPU determină o comandă pentru executarea proceselor sale programate. Având în vedere că n procesele care urmează să fie programate pe un procesor, câte planuri diferite sunt posibile? Dați o formulă în termeni de n.

Răspuns:

$$n! \text{ (n factorial} = n \times n - 1 \times n - 2 \times \dots \times 2 \times 1).$$

2. Explicați diferența dintre planificarea preemptivă și nonpreemptive.

Răspuns:

Programarea preemptivă permite întreruperea procesului în timpul executării acestuia, scoaterea CPU-ului și alocarea acestuia unui alt proces. Programarea fără previziune asigură faptul că un proces renunță la controlul CPU-ului numai atunci când acesta termină cu explozia actuală a CPU-ului.

3. Să presupunem că următoarele procese ajung să fie executate în momentele indicate. Fiecare proces va fi rulat pentru perioada de timp listată. Pentru a răspunde la întrebări, utilizați planificarea fără preempție și bazați toate deciziile pe informațiile pe care le aveți la momentul luării deciziei.

Process	Arrival Time	Burst Time
P_1	0.0	8
P_2	0.4	4
P_3	1.0	1

- a) Care este timpul mediu de reacție pentru aceste procese cu algoritmul de programare FCFS?
b) Care este timpul mediu de reacție pentru aceste procese cu algoritmul de programare SJF?
c) Algoritmul SJF ar trebui să îmbunătățească performanța, dar observați că am ales să executăm procesul P_1 la ora 0, deoarece nu știm că două procese mai scurte ar ajunge în curând. Stabiliti ce înseamnă timpul mediu de revenire dacă CPU este lăsat inactiv pentru prima unitate și apoi este programat programarea SJF. Amintiți-vă că procesele P_1 și P_2 sunt în aşteptare în acest timp de aşteptare, astfel încât timpul lor de aşteptare poate crește.

Acest algoritm ar putea fi cunoscut sub numele de planificare a cunoștințelor viitoare.

Răspuns:

- a) 10,53
b) 9,53
c) 6,86

Amintiți-vă că timpul de întoarcere este timpul de finalizare minus timpul de sosire, deci trebuie să scăpați timpul de sosire pentru a calcula orele de întoarcere. FCFS este de 11 dacă uități să scade timpul de sosire.

4. Ce avantaj există în a avea diferite dimensiuni de timp-cuantum la diferite niveluri ale unui sistem de aşteptare pe mai multe niveluri?

Răspuns:

Procesele care necesită o reparație mai frecventă, de exemplu, procese interactive cum ar fi editori, pot fi într-o coadă cu un quantum mic de timp. Procesele care nu necesită reparații frecvente pot fi într-o coadă cu un quantum mai mare, necesitând mai puține comutatoare de context pentru a finaliza procesarea și, astfel, pentru o utilizare mai eficientă a calculatorului.

5. Mulți algoritmi de programare CPU sunt parametrizați. De exemplu, algoritmul RR necesită un parametru pentru a indica fragmentul de timp. Cozi de feedback multi nivel necesită parametrii pentru a defini numărul de cozi, algoritmii de programare pentru fiecare coadă, criteriile utilizate pentru a muta procesele între cozile de aşteptare și aşa mai departe. Prin urmare, algoritmii sunt într-adevăr seturi de algoritmi (de exemplu, setul de algoritmi RR

pentru toate secțiunile de timp și aşa mai departe). Un set de algoritmi poate include un alt algoritm (de exemplu, algoritmul FCFS este algoritmul RR cu un quantum de timp infinit). Ce legătură (dacă există) există între următoarele perechi de seturi de algoritmi?

- a) Prioritate și SJF
- b) Cozi de feedback multi-nivel și FCFS
- c) Prioritate și FCFS
- d) RR și SJF

Răspuns:

a) Cea mai scurtă treabă are cea mai mare prioritate.

b) Cel mai mic nivel al MLFQ este FCFS.

c) FCFS acordă cea mai mare prioritate lucrului care a existat cel mai mult.

d) Nici unul.

6. Să presupunem că un algoritm de programare (la nivelul planificării CPU pe termen scurt) favorizează acele procese care au folosit cel mai puțin timp procesor în trecutul recent. De ce acest algoritm va favoriza programele legate de I/O și nu va afecta permanent programele legate de CPU?

Raspuns:

Va favoriza programele legate de I/O din cauza cererii relativ scurte de spargere a procesorului de către ei; cu toate acestea, programele legate de CPU nu vor dispărea de foame, deoarece programele legate de I/O vor renunța CPU relativ adesea la I/O.

7. Distingeți între programarea PCS și SCS.

Răspuns:

Planificarea PCS se face local procesului. Acesta este modul în care biblioteca de fișiere programează fire pe LWP-urile disponibile. Programarea SCS este situația în care sistemul de operare programează firele kernel-ului. Pe sistemele care utilizează fie multi-la-unul sau multi-la-multe, cele două modele de planificare sunt fundamental diferite. Pe sistemele care utilizează unul față de unul, PCS și SCS sunt aceleași.

8. Să presupunem că un sistem de operare atașează firele de nivel utilizator la kernel folosind modelul multi-to-many și că procesul de mapare se face prin utilizarea LWP-urilor. Mai mult, sistemul permite dezvoltatorilor de programe să creeze fire în timp real. Este necesar să legeți un thread în timp real unui LWP?

Răspuns:

Da, altfel un fir de utilizator ar putea fi nevoit să concureze pentru un LWP disponibil înainte de a fi programat. Prin legarea thread-ului utilizator la un LWP, nu există o latență în așteptarea unui LWP disponibil; thread-ul utilizator în timp real poate fi programat imediat.

9. Planificatorul UNIX tradițional impune o relație inversă între numerele de prioritate și priorități: cu cât este mai mare numărul, cu atât este mai mică prioritatea. Planificatorul recalculează prioritățile procesului o dată pe secundă utilizând următoarea funcție:

$$\text{Priority} = (\text{recenta utilizare CPU} / 2) + \text{bază}$$

unde baza = 60 și recenta utilizare a CPU se referă la o valoare care indică cât de des un proces a folosit CPU, deoarece prioritățile au fost recalculate ultima dată. Presupunem că recenta utilizare a CPU pentru procesul P1 este de 40, pentru procesul P2 este 18 și pentru procesul P3 este 10. Care vor fi noile priorități pentru aceste trei proceze atunci când

prioritățile sunt recalculate? Pe baza acestor informații, planificatorul UNIX tradițional ridică sau coboară prioritatea relativă a procesului legat de CPU?

Răspuns:

Prioritățile atribuite proceselor sunt 80, 69 și respectiv 65. Planificatorul scade prioritatea relativă a proceselor legate de CPU.

Part 3: Memory Management

7. Memory Management

- Denumiți două diferențe între adresele fizice și logice.

Răspuns:

O adresă logică nu se referă la adresa actuală; mai degrabă, se referă la o adresă abstractă într-un spațiu de adrese abstract. Contrastăți acest lucru cu o adresă fizică care se referă la o adresă fizică reală din memorie. O adresă logică este generată de CPU și este tradusă într-o adresă fizică de către unitatea de gestionare a memoriei (MMU). Prin urmare, adresele fizice sunt generate de MMU.

- Luați în considerare un sistem în care un program poate fi separat în două părți: cod și date. CPU-ul știe dacă dorește să primească o instrucțiune (preluarea instrucțiunilor) sau date (preluarea sau stocarea datelor). Prin urmare, sunt furnizate două perechi de registru de limită de bază: una pentru instrucțiuni și una pentru date. Perechea de registru pentru limita de instrucțiuni este automat citită, astfel încât programele pot fi distribuite între diferiți utilizatori. Discutați despre avantajele și dezavantajele acestei scheme.

Răspuns:

Principalul avantaj al acestei scheme este că este un mecanism eficient de partajare a codurilor și a datelor. De exemplu, numai o copie a unui editor sau a unui compilator trebuie păstrată în memorie, iar acest cod poate fi împărțit de toate procesele care au nevoie de acces la editorul sau codul compilatorului. Un alt avantaj este protejarea codului împotriva modificării eronate. Singurul dezavantaj este că codul și datele trebuie să fie separate, care de obicei se aderă într-un cod generat de compilator.

- De ce dimensiunile paginilor sunt întotdeauna de 2?

Răspuns:

Amintiți-vă că paginarea este implementată prin ruperea unei adrese într-o pagină și a unui număr de offset. Este cel mai eficient pentru a sparge adresa în biți de pagină X și biți Y offset, mai degrabă decât să efectuați aritmetică la adresa pentru a calcula numărul paginii și offsetul. Deoarece fiecare poziție de biți reprezintă o putere de 2, împărțirea unei adrese între biți duce la o dimensiune a paginii cu o putere de 2.

- Luați în considerare un spațiu logic de adrese de 64 de pagini de 1024 de cuvinte fiecare, mapate pe o memorie fizică de 32 de cadre.
 - Câți biți există în adresa logică?
 - Câți biți există în adresa fizică?

Răspuns:

- Adresa logică: 16 biți
- Adresa fizică: 15 biți

5. Care este efectul de a permite ca două intrări într-o tabelă de pagini să indice același cadru de pagină din memorie? Explicați modul în care acest efect ar putea fi utilizat pentru a reduce timpul necesar pentru a copia o cantitate mare de memorie dintr-un loc în altul. Ce efect ar avea actualizarea unor octeți pe o singură pagină pe cealaltă pagină?

Răspuns:

Prin permiterea a două intrări într-o tabelă de pagini pentru a indica același cadru de pagină din memorie, utilizatorii pot partaja codul și datele. Dacă codul este reentrant, spațiul de memorie poate fi salvat prin utilizarea în comun a programelor mari, cum ar fi editorii de text, compilatoarele și sistemele de baze de date. "Copierea" unor cantități mari de memorie ar putea fi efectuată prin faptul că diferite tabele de pagini indică aceeași locație de memorie. Cu toate acestea, împărtășirea codului sau a datelor nereținute înseamnă că orice utilizator care are acces la cod poate modifica acest cod și aceste modificări se vor reflecta în "copia" acestuia.

6. Descrieți un mecanism prin care un segment ar putea aparține spațiului de adrese al două procese diferite.

Răspuns:

7. Deoarece tabelele de segmente sunt o colecție de registre de limită de bază, segmentele pot fi partajate atunci când intrările din tabelul de segmente ale a două sarcini diferite indică aceeași locație fizică. Tabelele cu două segmente trebuie să aibă indicatori de bază identici, iar numărul segmentului partajat trebuie să fie același în cele două procese.

Partajarea segmentelor între procese fără a se cere ca aceștia să aibă același număr de segment este posibil într-un sistem de segmentare dinamic conectat.

- Definiți un sistem care permite legarea statică și partajarea segmentelor fără a cere ca numerele de segment să fie identice.
- Descrieți o schemă de paginare care permite partajarea paginilor fără a cere ca numerele paginilor să fie identice.

Răspuns:

Ambele probleme reduc la un program care poate să facă referire atât la propriul cod, cât și la datele sale, fără să știe numărul segmentului sau al paginii asociat cu adresa. MULTICS a rezolvat această problemă prin asocierea a patru registre cu fiecare proces. Un registru avea adresa segmentului curent de programe, celul avea o adresă de bază pentru stack, altul avea o adresă de bază pentru datele globale și aşa mai departe. Ideea este că toate referințele trebuie să fie indirecte printr-un registru care să corespundă segmentului curent sau numărului de pagină. Prin schimbarea acestor registre, același cod poate fi executat pentru diferite procese fără aceleași numere de pagină sau segmente.

8. În IBM / 370, protecția memoriei este asigurată prin utilizarea tastelor. O cheie este o cantitate de 4 biți. Fiecare bloc de memorie de 2K are o cheie (cheia de stocare) asociată cu ea. CPU are de asemenea o cheie (cheia de protecție) asociată cu aceasta. O operațiune de stocare este permisă numai dacă ambele chei sunt egale sau dacă este zero. Care dintre următoarele scheme de gestionare a memoriei ar putea fi utilizate cu succes cu acest hardware?
- Masina neteda
 - Sistem cu un singur utilizator
 - Multiprogramare cu un număr fix de procese
 - Multiprogramare cu un număr variabil de procese

- e) Paging
- f) segmentarea

Răspuns:

- a) Protecția nu este necesară, setați tasta de sistem la 0.
- b) Setați tasta de sistem la 0 când este în modul supervisor.
- c) Dimensiunile regiunilor trebuie să fie fixate în trepte de 2k octeți, alocarea cheii cu blocuri de memorie.
- d) La fel ca mai sus.
- e) Dimensiunile cadrelor trebuie să fie în trepte de 2k octeți, să aloce cheie cu pagini.
- f) Dimensiunile segmentului trebuie să fie în trepte de 2k octeți, să aloce cheie cu segmente.

8. Virtual Memory

1. În ce condiții apar defecțiuni de pagină? Descrieți acțiunile întreprinse de sistemul de operare atunci când apare o eroare de pagină.

Răspuns:

O eroare de pagină apare atunci când are loc accesul la o pagină care nu a fost introdusă în memoria principală. Sistemul de operare verifică accesul la memorie, întreruperea programului dacă este nevalidă. Dacă este valid, este amplasat un cadru liber și l/O este rugat să citească pagina necesară în cadru liber. După finalizarea intrărilor / ieșirilor, tabela de proces și tabela de pagini sunt actualizate și instrucțiunea este reluată.

2. Să presupunem că aveți un sir de referință pentru un proces cu m cadre (înțial toate goale). Sirul de referință al paginii are lungimea p; n apar numere distincte ale paginilor. Răspundeți la aceste întrebări pentru orice algoritmi de înlocuire a paginilor:
 - a) Care este limita inferioară a numărului de defecte de pagină?
 - b) Ce este o limită superioară a numărului de defecte de pagină?

Răspuns:

- a) n
- b) p

3. Luați în considerare tabelul de pagini prezentat în Figura 9.30 pentru un sistem cu adrese virtuale și fizice pe 12 biți și cu pagini de 256 octeți. Lista de cadre gratuite a paginilor este D, E, F (adică D este în fruntea listei, E este a doua și F este ultima). Conversia următoarelor adrese virtuale la adresele lor fizice echivalente în hexazecimal. Toate numerele sunt date în hexazecimal. (O bordură pentru un cadru de pagină indică faptul că pagina nu este în memorie.)

- 9EF
- 111
- 700
- OFF

Raspuns:

- 9E F - 0E F
- 111 - 211
- 700 - D00
- OF F – EFF

4. Luați în considerare următorii algoritmi de înlocuire a paginilor. Clasificați aceste algoritmi pe o scală de cinci puncte, de la "rău" la "perfect", în funcție de rata de eroare a paginilor. Separați acei algoritmi care suferă de anomalia lui Belady de la cei care nu.

- a) Înlocuirea LRU
- b) Înlocuirea FIFO
- c) Înlocuire optimă
- d) A doua înlocuire a şanselor

Raspuns:

<u>Rank</u>	<u>Algorithm</u>	<u>Suffer from Belady's anomaly</u>
1	Optimal	no
2	LRU	no
3	Second-chance	yes
4	FIFO	yes

5. Discutați despre suportul hardware necesar pentru a sprijini procesul de paginare a cererii.

Răspuns:

6. Pentru fiecare operație de acces la memorie, tabela de pagini trebuie consultată pentru a verifica dacă pagina respectivă este rezidentă sau nu și dacă programul are drepturi de citire sau scriere pentru a accesa pagina. Aceste verificări trebuie să fie efectuate în hardware. Un TLB ar putea servi ca o memorie cache și va îmbunătăți performanța operației de căutare.

Un sistem de operare acceptă o memorie virtuală paginată, folosind un procesor central cu un ciclu de 1 microsecundă. Costă o suplimentare de 1 microsecundă pentru a accesa o altă pagină decât cea actuală. Paginile au 1000 de cuvinte, iar dispozitivul de paginare este un tambur care se rotește la 3000 de rotații pe minut și transferă 1 milion de cuvinte pe secundă. Următoarele măsurători statistice au fost obținute de la sistem:

- 1% din instrucțiunile executate au accesat o altă pagină decât pagina curentă.
- Din instrucțiunile care au accesat altă pagină, 80% au accesat o pagină deja în memorie.
- Când a fost necesară o pagină nouă, pagina înlocuită a fost modificată la 50% din timp.

Calculați timpul de instruire efectiv pe acest sistem, presupunând că sistemul rulează doar un proces și că procesorul este inactiv în timpul transferurilor de cilindru.

Raspuns:

$$\begin{aligned}
 \text{effective access time} &= 0.99 \times (1 \mu\text{sec} + 0.008 \times (2 \mu\text{sec}) \\
 &\quad + 0.002 \times (10,000 \mu\text{sec} + 1,000 \mu\text{sec}) \\
 &\quad + 0.001 \times (10,000 \mu\text{sec} + 1,000 \mu\text{sec})) \\
 &= (0.99 + 0.016 + 22.0 + 11.0) \mu\text{sec} \\
 &= 34.0 \mu\text{sec}
 \end{aligned}$$

7. Luați în considerare matricea bidimensională A:

```
int A[][] = new int[100][100];
```

unde $A[0][0]$ este în locația 200 într-un sistem de memorie paginată cu pagini de dimensiunea 200. Un mic proces care manipulează matricea se găsește în pagina 0 (locurile 0 până la 199). Astfel, fiecare preluare a instrucțiunilor va fi de la pagina 0.

Pentru trei cadre de pagini, câte defecte de pagină sunt generate de următoarele bucle de inițializare a arrayului, folosind înlocuirea LRU și presupunând că cadrul de pagină 1 conține procesul, iar celelalte două sunt inițial goale?

- a.

```
for (int j = 0; j < 100; j++)
    for (int i = 0; i < 100; i++)
        A[i][j] = 0;
```
- b.

```
for (int i = 0; i < 100; i++)
    for (int j = 0; j < 100; j++)
        A[i][j] = 0;
```

Răspuns:

a) 5,000

b) 50

8. Luați în considerare sirul de referință al următoarei pagini:

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

Câte defecte de pagină ar apărea pentru următorii algoritmi de înlocuire, presupunând unul, doi, trei, patru, cinci, șase sau șapte cadre? Amintiți-vă că toate cadrele sunt inițial goale, astfel încât primele pagini unice vor costa fiecare câte o greșală.

- LRU replacement
- FIFO replacement
- Optimal replacement

Răspuns:

<u>Number of frames</u>	<u>LRU</u>	<u>FIFO</u>	<u>Optimal</u>
1	20	20	20
2	18	18	15
3	15	16	11
4	10	14	8
5	8	10	7
6	7	10	7
7	7	7	7

9. Să presupunem că doriți să utilizați un algoritm de paginare care necesită un bit de referință (cum ar fi înlocuirea a două șansă sau modelul de set de lucru), dar hardware-ul nu oferă unul. Schițați modul în care puteți simula un bit de referință, chiar dacă unul nu a fost furnizat de hardware sau explicați de ce nu este posibil să faceți acest lucru. Dacă este posibil, calculați costul.

Răspuns:

Puteți folosi un bit valid / nevalid susținut în hardware pentru a simula bitul de referință. Inițial setați bitul la invalid. La prima referință se generează o capcană către sistemul de operare. Sistemul de operare va seta un bit software la 1 și va reseta bitul valid / nevalid la valabil.

10. Ați proiectat un nou algoritm de înlocuire a paginii pe care îl considerați optim. În unele cazuri de test contorsionat, apare anomalia lui Belady. Este noul algoritm optim? Explică-ți răspunsul.

Răspuns:

Nu. Un algoritm optim nu va suferi de anomalia lui Belady, deoarece - prin definiție - un algoritm optimal înlocuiește pagina care nu va fi folosită pentru cea mai lungă perioadă de timp. Anomalia lui Belady apare atunci când un algoritm de plasare a paginii evacuează o pagină care va fi necesară în viitorul imediat. Un algoritm optim nu ar fi selectat o astfel de pagină.

11. Segmentarea este similară paginilor de paginare, însă folosește pagini cu dimensiuni variabile. Definiți doi algoritmi de înlocuire a segmentului pe baza schemelor de plasare a paginii FIFO și LRU. Rețineți că, deoarece segmentele nu au aceeași dimensiune, segmentul ales pentru a fi înlocuit poate să nu fie suficient de mare pentru a lăsa suficiente locații consecutive pentru segmentul necesar. Luați în considerare strategiile pentru sistemele în care segmentele nu pot fi mutate și cele pentru sistemele unde pot.

Răspuns:

- a) FIFO. Găsiți primul segment suficient de mare pentru a se potrivi segmentului de intrare. Dacă relocarea nu este posibilă și niciun segment nu este suficient de mare, selectați o combinație de segmente a căror amintiri sunt contighe, care sunt "mai apropiate de prima din listă" și care pot găzdui segmentul nou. Dacă este posibilă relocarea, rearanjați memoria astfel încât primele segmente N suficient de mari pentru segmentul de intrare să fie contighe în memorie. Adăugați în spațiul liber spațiu orice spațiu rămas în ambele cazuri.
- b) LRU. Selectați segmentul care nu a fost utilizat pentru cea mai lungă perioadă de timp și care este destul de mare, adăugând un spațiu rămas în lista spațiului liber. Dacă niciun segment nu este suficient de mare, selectați o combinație a celor mai vechi segmente care sunt contighe în memorie (dacă relocarea nu este disponibilă) și care sunt suficient de mari. Dacă este disponibilă o relocare, rearanjați cele mai vechi segmente N pentru a fi contighe în memorie și înlocuiți-le cu noul segment.

12. Luați în considerare un sistem informatic pe bază de cerere, unde gradul de multiprogramare este în prezent stabilit la patru. Sistemul a fost recent măsurat pentru a determina utilizarea procesorului și a discului de paginare. Rezultatele sunt una dintre următoarele alternative. Pentru fiecare caz, ce se întâmplă? Poate crește gradul de multiprogramare pentru a crește utilizarea procesorului? Ajută paginarea?

- a) Utilizarea CPU de 13%; utilizarea discului 97%
- b) CPU de utilizare 87 la sută; utilizare pe disc 3 procente
- c) Utilizarea CPU de 13%; utilizarea discului 3 procente

Răspuns

- a) Se întâmplă bătăi de cap.
- b) Utilizarea procesorului este suficient de mare pentru a lăsa lucrurile în pace și pentru a crește gradul de multiprogramare.
- c) Măriți gradul de multiprogramare.

13. Avem un sistem de operare pentru o mașină care folosește registre de bază și de limită, dar am modificat mașina pentru a furniza o tabelă de pagini. Pot fi configurate tabelele de pagini pentru a simula registrele de bază și de limită? Cum pot fi ei, sau de ce nu pot fi ei?

Răspuns:

tabela de pagini poate fi configurată pentru a simula registrele de bază și de limită, cu condiția ca memoria să fie alocată în segmente de dimensiune fixă. În acest fel, baza unui segment poate fi introdusă în tabela de pagini și bitul valid / nevalid utilizat pentru a indica acea parte a segmentului ca fiind rezidentă în memorie. Va exista o problemă cu fragmentarea internă.

Part 4: Storage Management

9. Mass-Storage Structure

1. Este programarea discului, alta decât planificarea FCFS, folosită într-un mediu pentru un singur utilizator? Explicați răspunsul.

Răspuns:

Într-un mediu pentru un singur utilizator, coada de intrare / ieșire este de obicei goală. Cererile, în general, provin dintr-un singur proces pentru un bloc sau pentru o succesiune de blocuri consecutive. În aceste cazuri, FCFS este o metodă economică de programare a discurilor. Dar LOOK este aproape la fel de ușor de programat și va oferi performanțe mult mai bune atunci când mai multe procese efectuează I / O concurență, cum ar fi atunci când un browser Web preia date în fundal în timp ce sistemul de operare este de paginare și o altă aplicație este activă în prim-plan.

2. Explicați de ce programarea SSTF tinde să favorizeze cilindrii de mijloc peste cilindrii cei mai interiori și cei mai exteriori.

Răspuns:

Centrul discului este locația având distanța medie cea mai mică față de toate celelalte piste. Astfel, capul discului tinde să se îndepărteze de marginile discului. Iată un alt mod de a gândi. Locația curentă a capului împarte cilindrii în două grupe. Dacă capul nu este în centrul discului și o nouă solicitare ajunge, noua solicitare este mai probabil să se afle în grupul care include centrul discului; astfel, capul este mai probabil să se miște în această direcție.

3. De ce nu este luată în considerare latența rotativă, de obicei, în programarea discurilor? Cum ați modifica SSTF, SCAN și C-SCAN pentru a include optimizarea latenței?

Răspuns:

Majoritatea discurilor nu își exportă informațiile despre poziția rotativă către gazdă. Chiar dacă au făcut-o, timpul pentru ca această informație să ajungă la programator ar fi supus unei imprecizii, iar timpul consumat de planificator este variabil, astfel încât informațiile despre poziția de rotație ar deveni incorecte. Mai mult, solicitările discului sunt de obicei date ca numere de blocuri logice, iar maparea între blocurile logice și locațiile fizice este foarte complexă.

4. De ce este important să echilibram sistemul de fișiere I / O între discuri și controlere pe un sistem într-un mediu multitasking?

Răspuns:

Un sistem poate funcționa numai la viteza celui mai lent gheare. Discurile sau controlerile de disc sunt adesea blocajele în sistemele moderne, deoarece performanțele lor individuale nu pot ține pasul cu cel al procesorului și al magistralei de sistem. Prin echilibrarea I / O între discuri și controlere, nici un disc individual, nici un controler nu este copleșit, astfel încât să se evite blocajul.

5. Care sunt compromisurile implicate în revederea paginilor de cod din sistemul de fișiere în comparație cu utilizarea spațiului de swap pentru a le stoca?

Răspuns:

Dacă paginile de cod sunt stocate în spațiul de swap, acestea pot fi transferate mai rapid în memoria principală (deoarece alocarea spațiului de transfer este reglată pentru o performanță mai rapidă decât alocarea generală a sistemului de fișiere). Folosirea spațiului swap poate necesita timp de pornire dacă paginile sunt copiate acolo la invocarea procesului, mai degrabă decât să fie pagerate pentru a schimba spațiul la cerere. De asemenea, trebuie alocat mai mult spațiu de swap dacă este folosit atât pentru cod cât și pentru paginile de date.

6. Există vreo modalitate de a implementa o stocare cu adevărat stabilă? Explică-ți răspunsul.

Răspuns:

7. Depozitarea cu adevărat stabilă nu va pierde niciodată date. Tehnica fundamentală pentru stocarea stabilă este păstrarea mai multor copii ale datelor, astfel încât dacă o copie este distrusă, o altă copie este încă disponibilă pentru utilizare. Dar pentru orice schemă, ne putem imagina un dezastru suficient de mare ca toate copile să fie distruse.

Se spune uneori că banda este un mediu de acces secvențial, în timp ce un disc magnetic este un mediu cu acces aleatoriu. De fapt, adevararea unui dispozitiv de stocare pentru acces aleatoriu depinde de dimensiunea transferului. Termenul rate de transfer în flux reprezintă rata pentru un transfer de date aflat în desfășurare, excluzând efectul latenței de acces. Prin contrast, rata de transfer efectivă este raportul dintre octeți totali pe secundă totală, inclusiv timpul de vârf, cum ar fi latența de acces.

Să presupunem că într-un computer cache-ul de nivel 2 are o latență de acces de 8 nanosecunde și o viteză de transfer de 800 megabiți pe secundă, memoria principală are o latență de acces de 60 nanosecunde și o viteză de transfer de 80 megaocteți pe secundă, discul magnetic are o latență de acces de 15 milisecunde și o rată de transfer de transmisie de 5 megabiți pe secundă, iar o unitate de bandă are o latență de acces de 60 de secunde și o viteză de transfer de 2 megabiți pe secundă .

- Accesul prin întâmplare determină scăderea ratei efective de transfer a unui dispozitiv, deoarece nu sunt transferate date în timpul accesului. Pentru discul descris, care este rata efectivă de transfer în cazul în care accesul mediu este urmat de un transfer de streaming de (1) 512 octeți, (2) 8 kilobiți, (3) 1 megabyte și (4) 16 megabytes?
- Utilizarea unui dispozitiv este raportul dintre rata de transfer efectivă și rata de transfer a fluxului. Calculați utilizarea unității de disc pentru fiecare din cele patru dimensiuni de transfer menționate în partea a.
- Să presupunem că o utilizare de 25% (sau mai mare) este considerată acceptabilă. Folosind cifrele de performanță date, calculați cea mai mică dimensiune de transfer pentru disc, care oferă o utilizare acceptabilă.
- Completați următoarea propoziție: Un disc este un dispozitiv de acces la întâmplare pentru transferuri mai mari decât octeți și este un dispozitiv secvențial de acces pentru transferuri mai mici.
- Calculează dimensiunile minime de transfer care oferă o utilizare acceptabilă pentru cache, memorie și bandă.
- Când este o bandă un dispozitiv cu acces aleator și când este un dispozitiv secvențial accesibil?

Răspuns:

- a) Pentru 512 octeți, rata efectivă de transfer se calculează după cum urmează. ETR = dimensiunea transferului / timpul de transfer. Dacă X este dimensiunea transferului, atunci timpul de transfer este $((X / STR) + \text{lățime})$

Transfer time is $15\text{ms} + (512\text{B}/5\text{MB per second}) = 15.0097\text{ms}$.
Effective transfer rate is therefore $512\text{B}/15.0097\text{ms} = 33.12 \text{ KB/sec}$.
ETR for 8KB = .47MB/sec.
ETR for 1MB = 4.65MB/sec.
ETR for 16MB = 4.98MB/sec.

- b. Utilization of the device for 512B = $33.12 \text{ KB/sec} / 5\text{MB/sec} = .0064 = .64$

For 8KB = 9.4%.
For 1MB = 93%.
For 16MB = 99.6%.

- c. Calculate $.25 = \text{ETR/STR}$, solving for transfer size X.

STR = 5MB, so $1.25\text{MB/S} = \text{ETR}$.
 $1.25\text{MB/S} * ((X/5) + .015) = X$.
 $.25X + .01875 = X$.
 $X = .025\text{MB}$.

- d. A disk is a random-access device for transfers larger than K bytes (where $K >$ disk block size), and is a sequential-access device for smaller transfers.

- e. Calculate minimum transfer size for acceptable utilization of cache memory:

STR = 800MB, ETR = 200, latency = $8 * 10^{-9}$.
 $200(X\text{MB}/800 + 8 * 10^{-9}) = X\text{MB}$.
 $.25X\text{MB} + 1600 * 10^{-9} = X\text{MB}$.
 $X = 2.24 \text{ bytes}$.

Calculate for memory:
STR = 80MB, ETR = 20, L = $60 * 10^{-9}$.
 $20(X\text{MB}/80 + 60 * 10^{-9}) = X\text{MB}$.
 $.25X\text{MB} + 1200 * 10^{-9} = X\text{MB}$.

$X = 1.68 \text{ bytes}$.

Calculate for tape:
STR = 2MB, ETR = .5, L = 60s.
 $.5(X\text{MB}/2 + 60) = X\text{MB}$.
 $.25X\text{MB} + 30 = X\text{MB}$.
 $X = 40\text{MB}$.

- f) Depinde de modul în care este folosit. Să presupunem că folosim banda pentru a restaura o copie de rezervă. În acest caz, banda funcționează ca un dispozitiv secvențial de accesare în care citim secvențial conținutul benzii. Ca un alt exemplu, să presupunem că folosim banda pentru a accesa o varietate de înregistrări stocate pe bandă. În acest caz, accesul la bandă este arbitrar și, prin urmare, este considerat întâmplător.

8. Ar putea o organizație de nivel RAID 1 să obțină o performanță mai bună pentru solicitările de citire decât o organizație de nivel 0 RAID (cu redundanță nerelevantă de date)? Dacă da, cum?

Răspuns:

Da, o organizație RAID de nivel 1 ar putea obține o performanță mai bună pentru solicitările de citire. Când se efectuează o operație de citire, un sistem RAID de nivel 1 poate decide care dintre cele două

copii ale blocului trebuie să fie accesată pentru a satisface cererea. Această alegere ar putea fi bazată pe locația actuală a capului de disc și ar putea, prin urmare, să conducă la optimizarea performanțelor, prin alegerea unui cap de disc care este mai aproape de datele căută.

10. File-System Interface

1. Unele sisteme elimină automat toate fișierele utilizatorilor atunci când un utilizator se deconectează sau se termină o activitate, cu excepția cazului în care utilizatorul solicită în mod explicit că acestea să fie păstrate; alte sisteme păstrează toate fișierele, cu excepția cazului în care utilizatorul le șterge în mod explicit. Discutați meritele relative ale fiecărei abordări.

Răspuns:

Ștergerea tuturor fișierelor care nu sunt salvate în mod specific de către utilizator are avantajul de a minimiza spațiul de fișier necesar pentru fiecare utilizator, prin faptul că nu salvează fișiere nedorite sau inutile. Salvarea tuturor fișierelor, cu excepția cazului în care sunt șterse în mod specific, este mai sigură pentru utilizator, deoarece nu este posibil să pierdeți fișierele din greșală, uitând să le salvați.

2. De ce unele sisteme urmăresc tipul unui fișier, în timp ce altele îl lasă pe utilizator, iar altele pur și simplu nu implementează mai multe tipuri de fișiere? Ce sistem este "mai bun?"

Răspuns:

Unele sisteme permit diferite operațiuni de fișiere pe baza tipului de fișier (de exemplu, un fișier ascii poate fi citit ca un flux în timp ce un fișier de bază de date poate fi citit printr-un index într-un bloc). Alte sisteme lasă o astfel de interpretare a datelor unui fișier în proces și nu oferă niciun ajutor în accesarea datelor. Metoda care este "mai bună" depinde de necesitățile proceselor din sistem și de cerințele pe care utilizatorii le plasează în sistemul de operare. Dacă un sistem rulează în cea mai mare parte aplicații de baze de date, poate fi mai eficient pentru sistemul de operare să implementeze un fișier de bază de date și să furnizeze operațiuni, mai degrabă decât să facă fiecare program să implementeze același lucru (posibil în diferite moduri). Pentru sistemele cu destinație generală, ar fi mai bine să implementați numai tipuri de fișiere de bază pentru a menține dimensiunea sistemului de operare mai mică și pentru a permite o libertate maximă proceselor din sistem.

3. În mod similar, unele sisteme acceptă multe tipuri de structuri pentru datele unui fișier, în timp ce altele suportă pur și simplu un flux de octeți. Care sunt avantajele și dezavantajele fiecărei abordări?

Răspuns:

Un avantaj al suportului de sistem al diferitelor structuri de fișiere este că suportul provine de la sistem; aplicațiile individuale nu sunt obligate să furnizeze suportul. În plus, dacă sistemul oferă suport pentru diferite structuri de fișiere, acesta poate implementa suportul probabil mai eficient decât o aplicație. Dezavantajul faptului că sistemul are suport pentru tipurile de fișiere definite este acela că mărește dimensiunea sistemului. În plus, este posibil ca aplicațiile care pot necesita diferite tipuri de fișiere decât cele furnizate de sistem să nu poată funcționa pe astfel de sisteme. O strategie alternativă este ca sistemul de operare să nu definească suport pentru structurile de fișiere și să trateze toate fișierele ca o serie de octeți. Aceasta este abordarea sistemelor UNIX. Avantajul acestei abordări este acela că simplifică suportul sistemului de operare pentru sistemele de fișiere, deoarece sistemul nu mai trebuie să furnizeze structura pentru diferite tipuri de fișiere. În plus, permite aplicațiilor să definească structuri de fișiere, reducând astfel situația în care un sistem nu poate furniza o definiție de fișier necesară unei anumite aplicații.

4. Puteți simula o structură de directoare pe mai multe niveluri cu o structură de directoare cu un singur nivel în care pot fi folosite nume lungi arbitrar? Dacă răspunsul dvs. este da, explicați cum puteți face acest lucru și contrastați această schemă cu schema de directoare pe mai multe niveluri. Dacă răspunsul dvs. este nu, explicați ce împiedică succesul simulării dvs. Cum s-ar schimba răspunsul dvs. dacă numele fișierelor ar fi limitate la șapte caractere?

Răspuns:

Dacă se pot utiliza nume lungi arbitrar, este posibilă simularea unei structuri de directoare pe mai multe niveluri. Acest lucru se poate face, de exemplu, folosind caracterul "." Pentru a indica sfârșitul unui subdirector. Astfel, de exemplu, numele jim.java.F1 specifică faptul că F1 este un fișier în java subdirector care, la rândul său, este în directorul rădăcină jim. Dacă numele fișierelor erau limitate la șapte caractere, atunci schema de mai sus nu a putut fi utilizată și, în general, răspunsul este nu. Următoarea abordare cea mai bună în această situație ar fi folosirea unui fișier specific ca tabel de simbol (director) pentru a mapa nume lungi arbitrar (cum ar fi jim.java.F1) în nume arbitrate mai scurte (cum ar fi XX00743), care sunt apoi utilizate pentru accesul real la fișiere.

5. Explicați scopul operațiunilor open () și close ().

Răspuns:

Scopul operațiilor open () și close () este:

- Operația open () informează sistemul că fișierul numit urmează să devină activ.
- Operația close () informează sistemul că fișierul numit nu mai este activ în utilizare de către utilizatorul care a emis operația de închidere.

6. În unele sisteme, un subdirector poate fi citit și scris de către un utilizator autorizat, la fel ca și fișierele obișnuite.
- a) Descrieți problemele de protecție care ar putea apărea.
 - b) Propuneți o schemă pentru a aborda fiecare dintre aceste probleme de protecție.

Răspuns:

- a) singură informație păstrată într-o intrare în director este locația fișierului. Dacă un utilizator ar putea modifica această locație, atunci ar putea accesa alte fișiere înfrângând schema de protecție a accesului.
- b) Nu permiteți utilizatorului să scrie direct pe subdirector. Mai degrabă, furnizați operațiunile de sistem pentru a face acest lucru.

7. Luați în considerare un sistem care suportă 5000 de utilizatori. Să presupunem că doriți să permiteți unui număr de 4.990 de utilizatori să acceseze un fișier.
- a) Cum ați specifica această schemă de protecție în UNIX?
 - b) Puteți sugera o altă schemă de protecție care poate fi utilizată mai eficient în acest scop decât schema oferită de UNIX?

Răspuns:

- a) Există două metode pentru a realiza acest lucru:
 - i. Creați o listă de control al accesului cu numele tuturor celor 4990 de utilizatori.
 - ii. Puneți acești 4990 de utilizatori într-un singur grup și setați accesul la grup în consecință. Această schemă nu poate fi întotdeauna implementată deoarece grupurile de utilizatori sunt restricționate de sistem.

- b) Accesul universal la fișiere se aplică tuturor utilizatorilor, cu excepția cazului în care numele lor apare în lista de control al accesului cu permisiune de acces diferită. Cu această schemă, pur și simplu puneți numele celor zece utilizatori rămași în lista de control al accesului, dar fără permisiuni de acces.
8. Cercetătorii au sugerat că, în loc să aibă o listă de acces asociată fiecărui fișier (specificând care dintre utilizatori pot accesa fișierul și cum), ar trebui să avem o listă de control a utilizatorului asociată cu fiecare utilizator (specificând fișierele pe care un utilizator le poate accesa și cum). Discutați despre meritele relative ale acestor două scheme.

Răspuns:

- Lista de control fișiere. Deoarece informațiile de control al accesului sunt concentrate într-un singur loc, este mai ușor să se schimbe informațiile de control al accesului și acest lucru necesită mai puțin spațiu.
- Lista de control al utilizatorului. Acest lucru necesită mai puțin aeriene la deschiderea unui fișier

11. File-System Implementation

1. Luați în considerare un fișier format în prezent din 100 de blocuri. Să presupunem că blocul de control al fișierelor (și blocul index, în cazul alocării indexate) este deja în memorie. Calculați câte operații de intrare / ieșire pe disc sunt necesare pentru strategiile de alocare continuă, asociate și indexate (cu un singur nivel) dacă, pentru un bloc, se mențin următoarele condiții. În cazul alocării contigue, presupuneți că nu există loc pentru a crește la început, dar nu există spațiu să crească la sfârșit. De asemenea, presupuneți că informațiile blocului care urmează să fie adăugate sunt stocate în memorie.
- a) Blocul este adăugat la început.
 - b) Blocul este adăugat în mijloc.
 - c) Blocul este adăugat la sfârșit.
 - d) Blocul este eliminat de la început.
 - e) Blocul este eliminat din mijloc.
 - f) Blocul este eliminat de la capăt.

Răspuns:

Rezultatele sunt:

	<u>Contiguous</u>	<u>Linked</u>	<u>Indexed</u>
a.	201	1	1
b.	101	52	1
c.	1	3	1
d.	198	1	0
e.	98	52	0
f.	0	100	0

2. Ce probleme ar putea apărea dacă un sistem a permis ca un sistem de fișiere să fie montat simultan la mai multe locații?

Răspuns:

Nu ar exista mai multe căi spre același fișier, ceea ce ar putea confunda utilizatorii sau să încurajeze greșelile (ștergerea unui fișier cu o singură cale șterge fișierul în toate celelalte căi).

3. De ce trebuie păstrată hartă bit pentru alocarea fișierelor pe spațiul de stocare în masă, mai degrabă decât în memoria principală?

Răspuns:

În cazul unui accident de sistem (eșec de memorie), lista de spațiu liber nu s-ar pierde aşa cum ar fi dacă hartă bit ar fi fost stocată în memoria principală

4. Luați în considerare un sistem care sprijină strategiile de alocare contiguă, legată și indexată. Ce criterii ar trebui să se folosească pentru a decide ce strategie este utilizată cel mai bine pentru un anumit dosar?

Răspuns:

- Fișierul contiguu-if este de obicei accesat secvențial, dacă fișierul este relativ mic.
 - Fișierul Linked-if este mare și de obicei accesat secvențial.
 - Fișier indexat-if este mare și de obicei accesat la întâmplare
5. O problemă cu alocarea contiguă este că utilizatorul trebuie să prealocheze spațiu suficient pentru fiecare fișier. Dacă fișierul devine mai mare decât spațiul alocat pentru acesta, trebuie luate măsuri speciale. O soluție la această problemă este definirea unei structuri de fișiere constând dintr-o zonă inițială contiguă (cu o dimensiune specificată). Dacă această zonă este umplută, sistemul de operare definește automat o zonă de supracurent care este legată de zona contiguă inițială. Dacă este umplută suprafața de supraînăltare, este alocată o altă zonă de suprapunere. Comparați această implementare a unui fișier cu implementările standard contigüe și legate.

Răspuns:

Această metodă necesită mai mult aeriene decât alocarea standard contiguă. Este nevoie de mai puțin aeriene decât alocarea standard asociată.

6. Cum ajută cache-urile să îmbunătățească performanța? De ce sistemele nu utilizează cache-uri mai mari sau mai mari dacă sunt atât de utile?

Răspuns:

Cache-urile permit componentelor de viteze diferite să comunice mai eficient prin stocarea temporară a datelor de pe dispozitivul mai lent într-un dispozitiv mai rapid (memoria cache). Cache-urile sunt, aproape prin definiție, mai scumpe decât dispozitivul pentru care sunt stocate cache-urile, astfel încât creșterea numărului sau mărimei cache-urilor ar crește costul sistemului.

7. De ce este avantajos pentru utilizatorul ca un sistem de operare să-și aloce dinamic tabelele interne? Care sunt sancțiunile sistemului de operare pentru a face acest lucru?

Răspuns:

Tabelele dinamice permit o mai mare flexibilitate în creșterea utilizării sistemului - tabelele nu sunt depășite niciodată, evitând limitele de utilizare artificială. Din păcate, structurile și codul kernel-ului sunt mai complicate, deci există un potențial mai mare pentru bug-uri. Utilizarea unei singure resurse poate duce la eliminarea mai multor resurse ale sistemului (prin creșterea pentru a satisface cererile) decât la tabelele statice.

8. Explicați modul în care stratul VFS permite unui sistem de operare să suporte cu ușurință mai multe tipuri de sisteme de fișiere.



Răspuns:

VFS introduce un strat de indirecție în implementarea sistemului de fișiere. În multe privințe, este similară cu tehniciile de programare orientate pe obiecte. Apelurile de sistem pot fi efectuate în mod generic (independent de tipul de sistem de fișiere). fiecare tip de sistem de fișiere oferă funcțiile sale de apeluri și structuri de date în stratul VFS. Un apel sistem este tradus în funcțiile specifice adecvate pentru sistemul de fișiere întă din stratul VFS. Programul de asteptare nu are cod specific pentru sistemul de fișiere, iar nivelurile superioare ale structurilor de apeluri de sistem sunt, de asemenea, independente de sistemul de fișiere. Traducerea la nivelul VFS transformă aceste apeluri generice în operațiuni specifice sistemului de fișiere.

12. I/O Systems

- Există trei avantaje de a pune funcționalitatea într-un controler de dispozitiv, mai degrabă decât în kernel. Există trei dezavantaje.

Răspuns:

Trei avantaje:

- Bug-urile sunt mai puțin susceptibile de a provoca un accident de sistem de operare
- Performanța poate fi îmbunătățită utilizând algoritmi dedicați și algoritmi codați greu
- Kernelul este simplificat prin mutarea algoritmilor din acesta.

Trei dezavantaje:

- Bug-urile sunt mai greu de rezolvat - este nevoie de o nouă versiune firmware sau de hardware nou
 - Îmbunătățirea algoritmilor necesită, de asemenea, o actualizare hardware, mai degrabă decât o actualizare a kernel-ului sau a driver-ului de dispozitiv
 - Algoritmii integrați ar putea intra în conflict cu utilizarea aplicației de către dispozitiv, cauzând scăderea performanței.
- Exemplu de strângere de mâna în Secțiunea 13.2 a folosit 2 biți: un bit ocupat și un bit de comandă. Este posibil să punem în aplicare această strângere de mâna cu un singur bit? Dacă este, descrieți protocolul. Dacă nu, explicați de ce un bit este insuficient.

Răspuns:

Este posibil, folosind următorul algoritm. Să presupunem că pur și simplu folosim bitul ocupat (sau bitul pregătit pentru comandă, acest răspuns fiind același indiferent). Când bitul este opriț, controlerul este inactiv. Gazda scrie datele de ieșire și stabileste bitul pentru a semnala că o operație este gata (echivalentul setării bitului pregătit pentru comandă). Când controlerul este terminat, acesta șterge bitul ocupat. Apoi, gazda inițiază următoarea operație. Această soluție cere ca atât gazda cât și controlerul să aibă acces la citirea și scrierea aceluiasi bit, ceea ce poate complica circuitele și poate crește costul controlerului.

- De ce ar putea un sistem să utilizeze I / O cu intrerupere pentru a gestiona un singur port serial și intrare / ieșire de sondaj pentru a gestiona un procesor frontal, cum ar fi un concentrator de terminale?

Răspuns:

Interogarea poate fi mai eficientă decât I / O cu intreruperi. Acesta este cazul când I / O este frecvent și de scurtă durată. Chiar dacă un singur port serial va efectua I / O relativ rar și ar trebui să utilizeze astfel intreruperi, o colecție de porturi seriale cum ar fi cele dintr-un concentrator terminal poate

produce o mulțime de operații I / O scurte și întreruperea pentru fiecare poate crea sarcină mare pe sistem. O buclă de sondare bine programată ar putea să atenuze acea încărcătură fără a pierde multe resurse prin looping fără nevoie de I / O.

4. Interogarea pentru o completare I / O poate duce la pierderea unui număr mare de cicluri de CPU dacă procesorul repetă o buclă de aşteptare ocupată de mai multe ori înainte de finalizarea intrărilor / ieșirilor. Dar, dacă dispozitivul I / O este gata pentru serviciu, interogarea poate fi mult mai eficientă decât capturarea și dispecerarea unei întreruperi. Descrieți o strategie hibridă care combină interogarea, dormitul și întreruperea pentru serviciul dispozitivelor I / O. Pentru fiecare dintre aceste trei strategii (polling pur, întreruperi pure, hibrid), descrieți un mediu de calcul în care această strategie este mai eficientă decât oricare dintre celelalte.

Răspuns:

O abordare hibridă ar putea comuta între interogare și întreruperi, în funcție de lungimea aşteptării operației I / O. De exemplu, am putea sondaj și loop N de ori, și dacă dispozitivul este încă ocupat la $N + 1$, am putea seta o întrerupere și somn. Această abordare ar evita ciclurile lungi de aşteptare ocupate. Această metodă ar fi cea mai bună pentru orele foarte ocupate sau foarte scurte. Ar fi ineficient ca I / O să se termine la $N + T$ (unde T este un număr redus de cicluri) datorită costurilor generale ale sondajului plus instalarea și întreruperea capturilor. Ancheta pură este cea mai bună, cu perioade foarte scurte de aşteptare. Întreruperile sunt cele mai bune cu timpuri de aşteptare cunoscute.

5. Cum DMA mărește concurența sistemului? Cum complică proiectarea hardware?

Răspuns:

DMA mărește concurența sistemului, permitând procesorului să efectueze sarcini în timp ce sistemul DMA transferă date prin intermediul sistemului și a magistralelor de memorie. Proiectarea hardware-ului este complicată deoarece controlerul DMA trebuie să fie integrat în sistem, iar sistemul trebuie să permită controlerului DMA să fie un maestru de magistrală. Furtul ciclului poate fi de asemenea necesar pentru a permite procesorului și controlerului DMA să împărtășească utilizarea busului de memorie.

6. De ce este important să se mărească viteza sistem-bus și a dispozitivului ca viteza procesorului crește?

Răspuns:

Luați în considerare un sistem care efectuează 50% I / O și 50% calculează. Duharea performanțelor CPU pe acest sistem ar crește performanța sistemului total cu doar 50%. Duharea ambelor aspecte ale sistemului ar spori performanța cu 100%. În general, este important să eliminați blocajul actual al sistemului și să creșteți performanța generală a sistemului, în loc să creșteți orbește performanțele componentelor individuale ale sistemului.

7. Distingeți între un driver STREAMS și un modul STREAMS.

Răspuns:

Driverul STREAMS controlează un dispozitiv fizic care ar putea fi implicat într-o operațiune STREAMS. Modulul STREAMS modifică fluxul de date între cap (interfață utilizator) și driver.