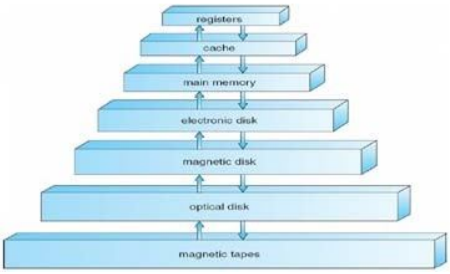
Study Guide for *Operating Systems*

Unit.1 - Introduction

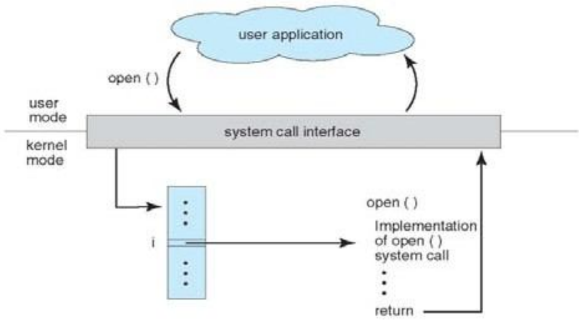
- An OS is a program that acts as an intermediary between a user of a computer and the computer hardware
- Goals: Execute user programs, make the comp. system easy to use, utilize hardware efficiently
- Computer system: Hardware ↔ OS ↔ Applications ↔ Users (↔ = 'uses')
- OS is:
    - Resource allocator: 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

OS Structures

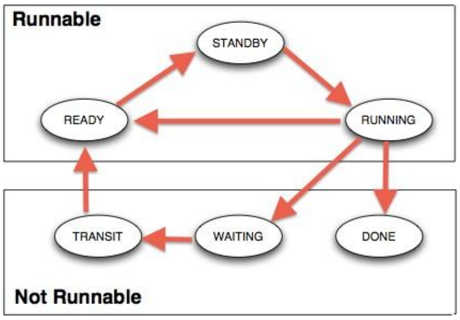
- User Interface (UI) – Can be Command-Line (CLI) or Graphics User Interface (GUI) or Batch
  - These allow for the user to interact with the system services via system calls (typically written in C/C++)
- Other system services that a helpful to the user include: program execution, I/O operations, file-system 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

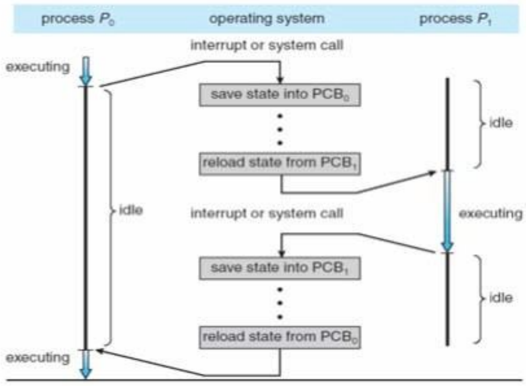


- Process control system calls include: end, abort, load, execute, create/terminate process, wait, allocate/free memory
- File management system calls include: create/delete file, open/close file, read, write, get/set attributes
- Device management system calls: request/release device, read, write, logically 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

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



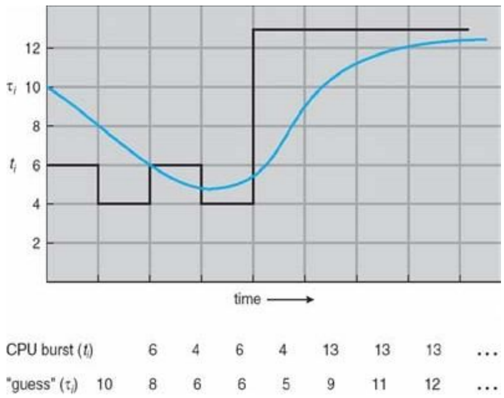


Unit 2 – Threads

- 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

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. α, $0 \le \alpha \le 1$ (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 quantums can result in large amounts of context switches
  - Time quantum should be chosen so that 80% of processes have shorter burst times that 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 = avg queue length; W = avg waiting time in queue; λ = 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

Unit 3 – 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);
```

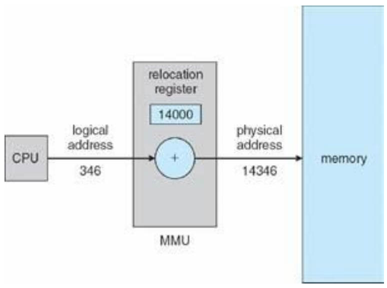
- Semaphore: Synchronization tool that does not require busy waiting
  - Standard operations: wait() and signal() ← these are the only operations that can access semaphore S
  - Can have counting (unrestricted range) and binary (0 or 1) semaphores
- Deadlock: Two or more processes are waiting indefinitely 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)

Deadlocks

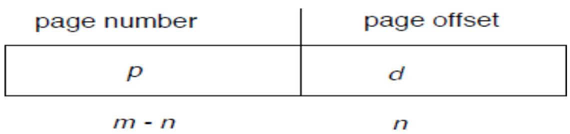
- Deadlock Characteristics: deadlock can occur if these conditions hold simultaneously
  - Mutual Exclusion: only one process at a time can use a resource
  - Hold and Wait: process holding one resource is waiting to acquire resource held by another process
  - No Preemption: a resource can be released 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

Unit 4 – Memory Management

- 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 copes 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



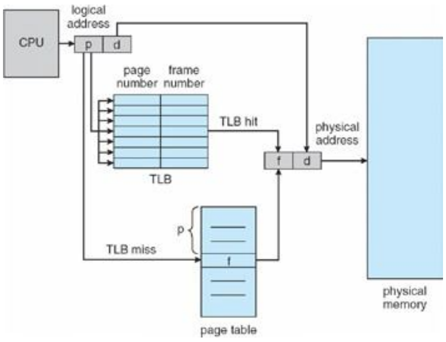
- Solutions to Dynamic Storage-Allocation Problem:
  - First-fit: allocate the first hole that is big enough
  - Best-fit: allocate the smallest hole that is big enough (must search entire list) → smallest leftover hole
  - Worst-fit: allocate the largest hole (search entire list) → largest leftover 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



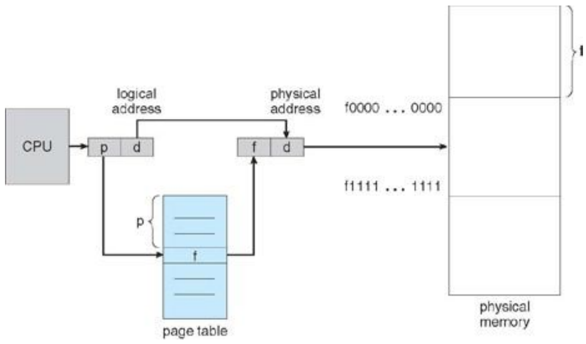
For given logical address space 2^m and page size 2^n

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

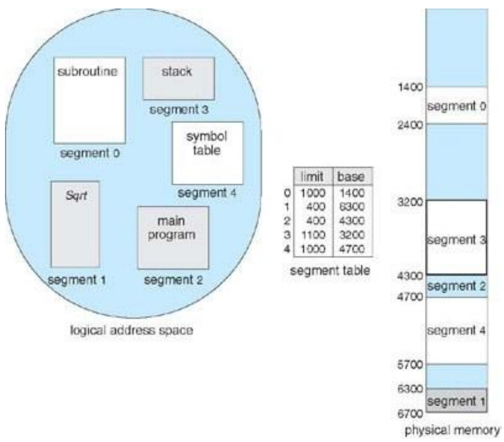


Paging with TLB

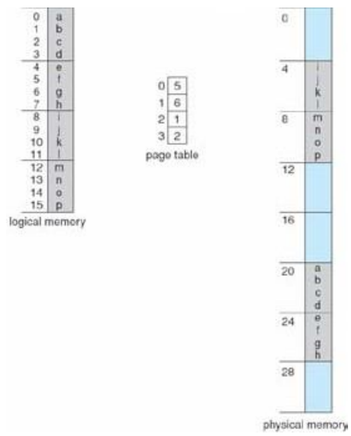


Paging without TLB

- 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



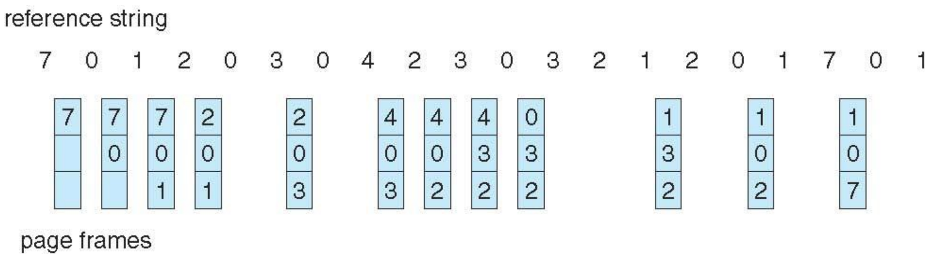
Segmentation example



Page table example

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)*memory access + p*(page fault overhead + swap page out + swap page in + 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)



- 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, $S = \Sigma s_i$, 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

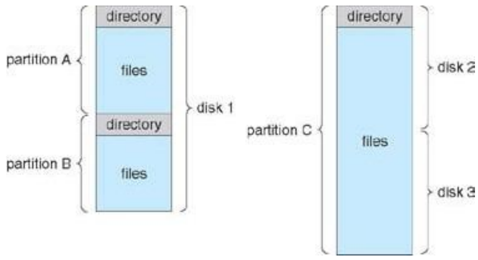
in memory

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

Unit 5 – File-System Interface

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, 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	arc, 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

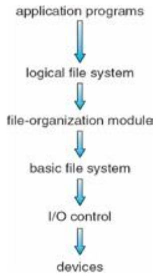
- File – Uniform logical view of information storage (no matter the medium)
  - Mapped onto physical devices (usually nonvolatile)
  - Smallest allotment of nameable storage
  - Types: Data (numeric, character, binary), Program, Free form, Structured
  - Structure decided by OS and/or program/programmer
- 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 knows as minidisks, slices



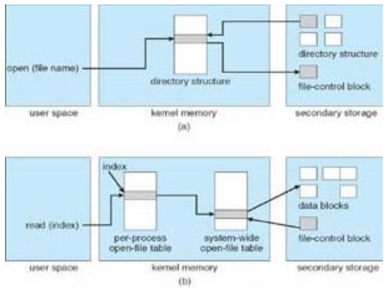
File-System Organization

- 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

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

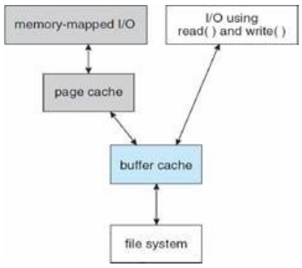


(a) open() (b) read()

- 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