ACMT Group of Colleges

Polytechnic- 2rd Year/ 4th Semester

Operating Systems Notes

By, Shivani Gupta (CS Department)

UNIT -1
COMPUTER SYSTEM AND OPERATING SYSTEM OVERVIEW

OVER VIEW OF OPERATING SYSTEM

What is an Operating System?

A program that acts as an intermediary between a user of a computer and the computer hardwareOperating system goals:

- Execute user programs and make solving
- user problems easierMake the computer system convenient to use

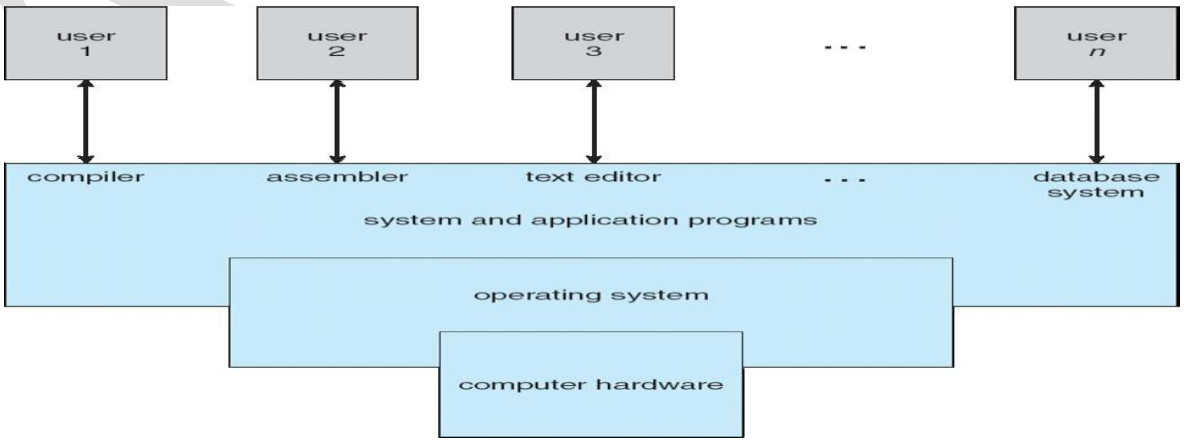
Use the computer hardware in an efficient manner

Computer System Structure

Computer system can be divided into four components • Hardware – provides basic computing resources

- CPU, memory, I/O devices •
- Operating system
  - Controls and coordinates use of hardware among various applications and users
- Application programs – define the ways in which the system resources are used to solve the computingproblems of the users
  - Word processors, compilers, web browsers, database
- systems, video gamesUsers
  - People, machines, other computers

Four Components of a Computer System



Operating System Definition

- OS is a **resource allocator**
- Manages all resources
- Decides between conflicting requests for efficient
- and fair resource useOS is a **control program**
- Controls execution of programs to prevent errors and improper use of the computerNo universally accepted definition

  Everything a vendor ships when you order an operating system" is good approximationBut varies wildly
- "The one program running at all times on the computer" is the **kernel.** Everything else is either asystem program (ships with the operating system) or an application program

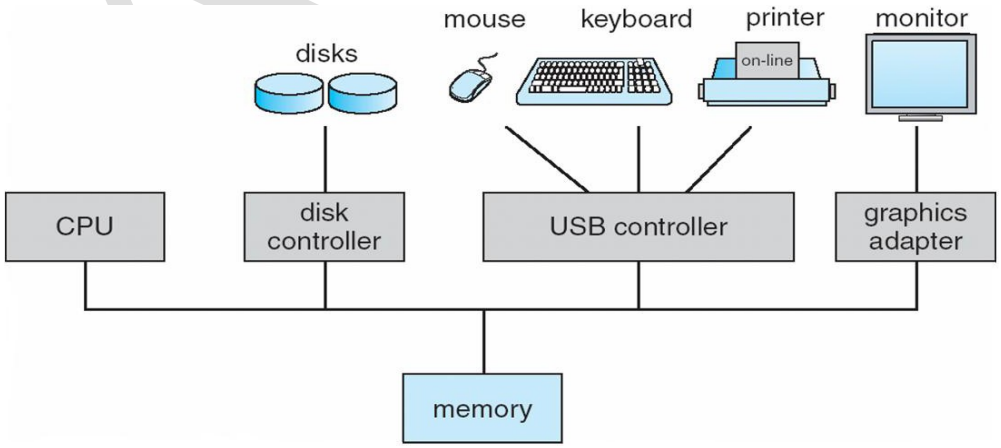
Computer Startup

- **bootstrap program** is loaded at power-up or reboot
- Typically stored in ROM or EPROM, generally known as **firmware**
- Initializes all aspects of system

  Loads operating system kernel and starts execution

Computer System Organization

- Computer-system operation
- One or more CPUs, device controllers connect through common bus providing access to shared memoryConcurrent execution of CPUs and devices competing for memory cycles



Computer-System Operation

- I/O devices and the CPU can execute concurrently
- Each device controller is in charge of a
- particular device typeEach device
- controller has a local buffer
- CPU moves data from/to main memory to/from local buffersI/O is from the device to local buffer of controller
  Device controller informs CPU that it has finished its operation by causing An *interrupt*

Common Functions of Interrupts

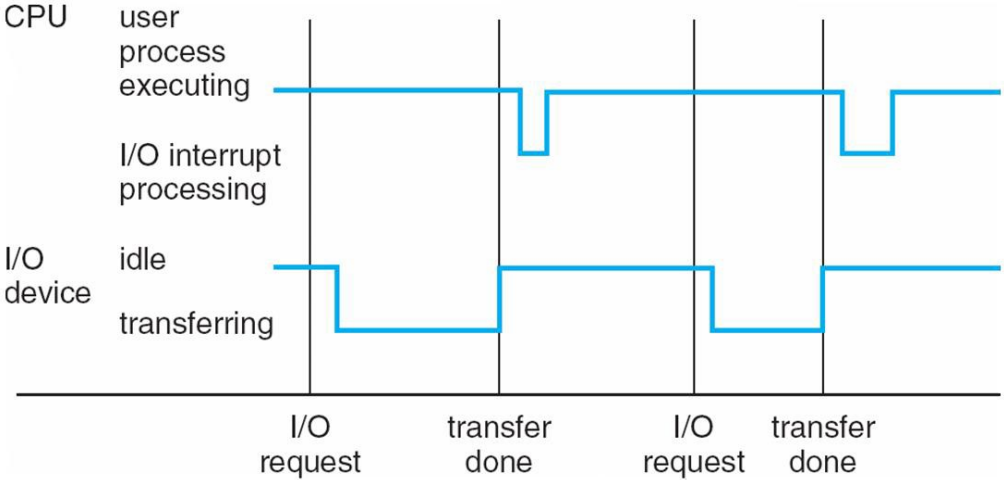
- Interrupt transfers control to the interrupt service routine generally, through the **interrupt vector**, whichcontains the addresses of all the
- service routines
- Interrupt architecture must save the address of the interrupted instruction
  Incoming interrupts are *disabled* while another interrupt is being processed to prevent a *lost interrupt*nA*trap* is a software-generated interrupt caused either by an error or a user request
- An operating system is

interrupt drivenInterrupt

- **Handling**
- The operating system preserves the state of the CPU by storing registers and the program counterDetermines which type of interrupt has occurred:
  polling

- **vectored** interrupt system
- Separate segments of code determine what action should be taken for each type of interrupt

Interrupt Timeline



I/O Structure

- After I/O starts, control returns to user program only
- upon I/O completion Wait instruction idles the CPU
- until the next interrupt
- Wait loop (contention for memory access)
- At most one I/O request is outstanding at a time, no
- simultaneous I/O processing After I/O starts, control returns to user program without waiting for I/O completion **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** Operating system indexes into I/O device table to determine device status and to modify table entry to include interrupt

Direct Memory Access Structure

- Used for high-speed I/O devices able to transmit information at close to memory speeds

  Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention
- Only one interrupt is generated per block, rather than the one interrupt per byte

Storage Structure

- Main memory – only large storage media that the CPU can access directly
- Secondary storage – extension of main memory that provides large
- nonvolatile storage capacity Magnetic disks – rigid metal or glass
- platters covered with magnetic recording material

  Disk surface is logically divided into **tracks**, which are subdivided into **sectors**

  The **disk controller** determines the logical interaction between the device and the computer

Storage Hierarchy

- Storage systems organized
- in hierarchy Speed
- Cost

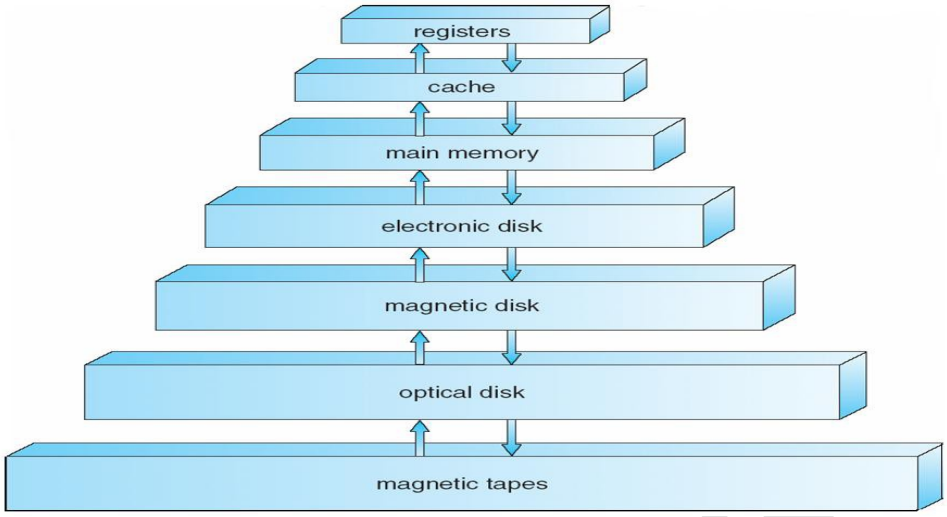
  Vol

  atilit

  y

Caching – copying information into faster storage system; main memory can be viewed as a last *cache* for

secondary storage



Caching

- Important principle, performed at many levels in a computer (in hardware,
- operating system, software)Information in use copied from slower to
- faster storage temporarily
- Faster storage (cache) checked first to determine if
- information is thereIf it is, information used
- directly from the cache (fast)
  If not, data copied to cache and
  used there Cache smaller than
  storage being cached Cache
  management important design
  problemCache size and
  replacement policy

Computer-System Architecture

- Most systems use a single general-purpose processor
- (PDAs through mainframes)Most systems have special-
- purpose processors as well
  Multiprocessors systems growing in
  use and importanceAlso known as
  parallel systems, tightly-coupled

systems

Advantages include

1. Increased throughput
2. Economy of scale
3. Increased reliability – graceful degradation or fault tolerance

Two types

1. Asymmetric Multiprocessing
2. Symmetric Multiprocessing

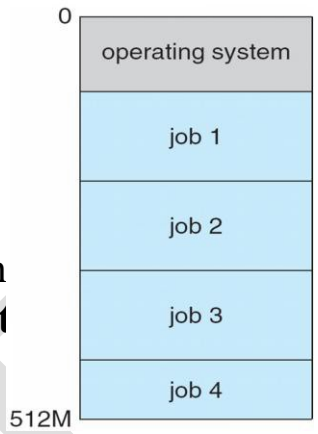
Clustered Systems

- Like multiprocessor systems, but multiple systems working together
- Usually sharing storage via a storage-area network (SAN)
- Provides a high-availability service which survives failures
  - Asymmetric clustering has one machine in hot-standby mode
  - Symmetric clustering has multiple nodes running applications, monitoring each other
- Some clusters are for high-performance computing (HPC)
  - Applications must be written to use parallelization

Operating System Structure

- **Multiprogramming** needed for efficiency
- Single user cannot keep CPU and I/O devices busy at all times
- Multiprogramming organizes jobs (code and data) so CPU always has one to Execute
- A subset of total jobs in system is kept in memory
- One job selected and run via **job scheduling**
- When it has to wait (for I/O for example), OS switches to another job
- **Timesharing (multitasking)** is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing

Response time should be < 1 second
Each user has at least one program executing in memory [**process** If several jobs ready to run at the same time [**CPU scheduling**
If processes don't fit in memory, **swapping** moves them in and out to run **Virtual memory** allows execution of processes not completely in memory **Memory Layout for Multiprogrammed System**



Operating-System Operations

- Interrupt driven by hardware
- Software error or request creates **exception** or **trap**
- Division by zero, request for operating system service
- Other process problems include infinite loop, processes modifying each
- Other or the operating system

Dual-mode operation allows OS to protect itself and other system components
User mode and
kernel mode Mode
bit provided by
hardware

- Provides ability to distinguish when system is running
- user code or kernel code Some instructions designated as **privileged**, only executable in kernel mode System call changes mode to kernel, return from call resets it to user

Transition from User to Kernel Mode

- Timer to prevent infinite loop / process
- hogging resources Set interrupt after
- specific period
- Operating system decrements counter
  When counter zero generate an interrupt
  Set up before scheduling process to regain control or terminate program that exceeds allotted time

Unit- 1

OPERATING SYSTEM FUNCTIONS

Process Management

- A process is a program in execution. It is a unit of work within the system. Program is a *passive entity*, process is an *active entity*.
- Process needs resources to
- accomplish its task CPU,
- memory, I/O, files
- Initialization data
- Process termination requires reclaim of any reusable resources
- Single-threaded process has one **program counter** specifying location of next instruction to execute Process executes instructions sequentially, one at a time, until completion
  Multi-threaded process has one program counter per thread
  Typically system has many processes, some user, some operating system running concurrently on one or more CPUs
- Concurrency by multiplexing the CPUs among the processes / threads
- **Memory management activities**
- Keeping track of which parts of memory are currently being
- used and by whom Deciding which processes (or parts thereof) and data to move into and out of memory Allocating and deallocating memory space as needed

Storage Management

- OS provides uniform, logical view of
- information storage Abstracts physical
- properties to logical storage unit - **file**
- Each medium is controlled by device (i.e., disk drive, tape drive)
- Varying properties include access speed, capacity, data-transfer rate,
- access method (sequential or random)
- File-System management
- Files usually organized into directories
- Access control on most systems to determine who can access what
  OS activities include

Mass-Storage Management

- Usually disks used to store data that does not fit in main memory or data that must be kept for a "long"period of time
  Proper management is of central importance
  Entire speed of computer operation hinges on disk subsystem and its algorithms

I/O Subsystem

- One purpose of OS is to hide peculiarities of hardware
- devices from the userI/O subsystem responsible for
  Memory management of I/O including buffering (storing data temporarily while it is being transferred),caching (storing parts of data in faster storage
- for performance), spooling (the overlapping of output of one job with
- input of other jobs)
  General device-driver
  interface Drivers for
  specific hardware
  devices

Protection and Security

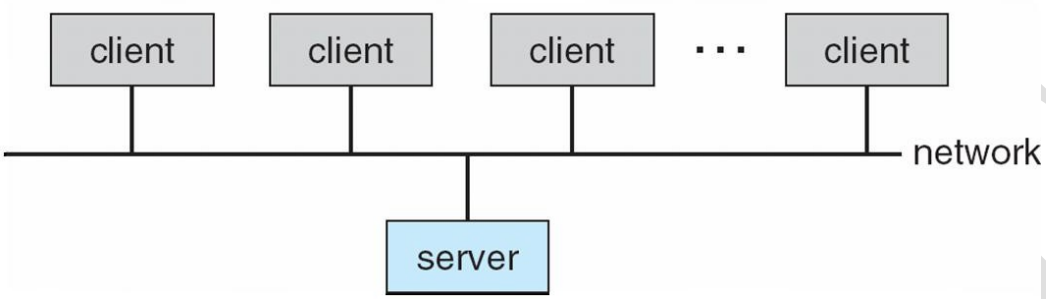
Protection – any mechanism for controlling access of processes or users to resources defined by the OS

Security – defense of the system against internal and external attacks

- Huge range, including denial-of-service, worms, viruses,
- identity theft, theft of serviceSystems generally first
- distinguish among users, to determine who can do what
- User identities (**user IDs**, security IDs) include name and associated number, one per userUser ID then associated with all files, processes of that user to determine access control
  Group identifier (**group ID**) allows set of users to be defined and controls managed, then also associatedwith each process, file
- **Privilege escalation** allows user to change to effective ID with more rights
- Dumb terminals supplanted by smart PCs
- Many systems now **servers**, responding to requests

generated by **clients Compute-server** provides an

interface to client to request services (i.e. database) **File-server** provides interface for clients to store and retrieve files



Peer-to-Peer Computing

- Another model of distributed system
- P2P does not distinguish
- clients and servers Instead all
- nodes are considered peers
  May each act as client,
  server or both Node must
  join P2P network
    - Registers its service with central lookup service on network, or
    - Broadcast request for service and respond to requests for service via **discovery protocol**
- Examples include *Napster* and *Gnutella*

Web-Based Computing

- Web has become
- ubiquitous PCs
- most prevalent
- devices
  More devices becoming networked to allow web access
  New category of devices to manage web traffic among similar servers: **load balancers**
  Use of operating systems like Windows 95, client-side, have evolved into Linux and Windows XP, which can be clients and servers

Open-Source Operating Systems

- Operating systems made available in source-code format rather
-
-

than just binary closed-sourceCounter to the copy protection and Digital Rights Management (DRM) movement
Started by Free Software Foundation (FSF), which has "copyleft" GNU Public License (GPL)Examples include GNU/Linux, BSD UNIX (including core of Mac OS X), and Sun Solaris

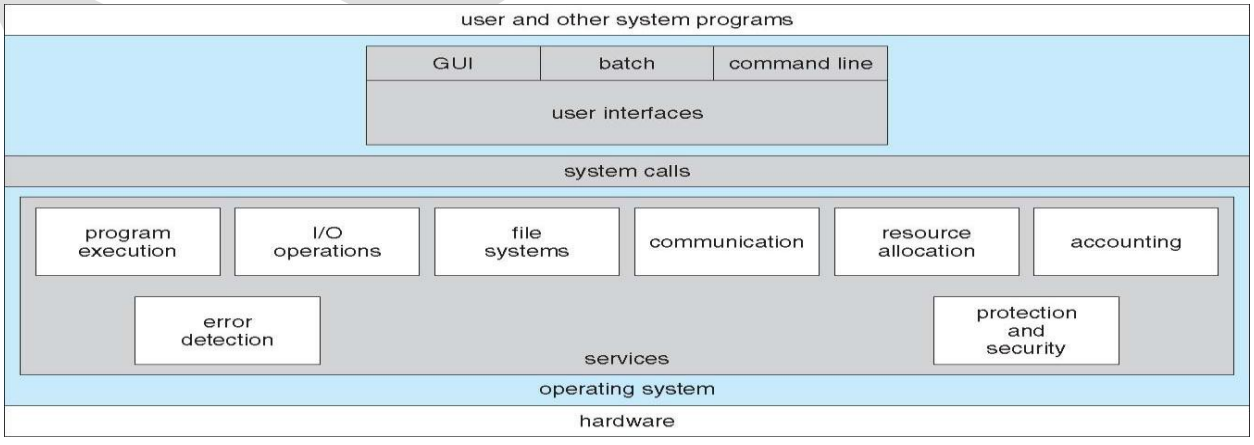
Operating System Services

- One set of operating-system services provides functions
- that are helpful to the user:User interface - Almost all
- operating systems have a user interface (UI)
  - Varies between Command-Line (CLI), Graphics User Interface (GUI), Batch

- Program execution - The system must be able to load a program into memory and to run that program,end execution, either normally or abnormally (indicating error)
- I/O operations - A running program may require I/O, which may involve a file or an I/O device

- File-system manipulation - The file system is of particular interest. Obviously, programs need to readand write files and directories, create and delete them, search them, list file Information, permission management.

A View of Operating System Services



Operating System Services

- One set of operating-system services provides functions that are helpful to the user

Communications – Processes may exchange information, on the same computer or between computers over a network Communications may be via shared memory or through message passing (packets moved by the OS)

- Error detection – OS needs to be constantly aware of possible errors May occur in the CPU and memory hardware, in I/O devices, in user program For each type of error, OS should take the appropriate action to ensure correct and consistent computing Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system

- Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing
- **Resource allocation -** When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
- Many types of resources - Some (such as CPU cycles, main memory, and file storage) may have special allocation code, others (such as I/O devices)
- may have general request and release code

Accounting - To keep track of which users use how much and what kinds of computer resources **Protection and security -** The owners of information stored in a multiuser or networked computer system may want

- to control use of that information, concurrent processes should not
- interfere with each other

Protection involves ensuring that all access to system resources is controlled **Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts

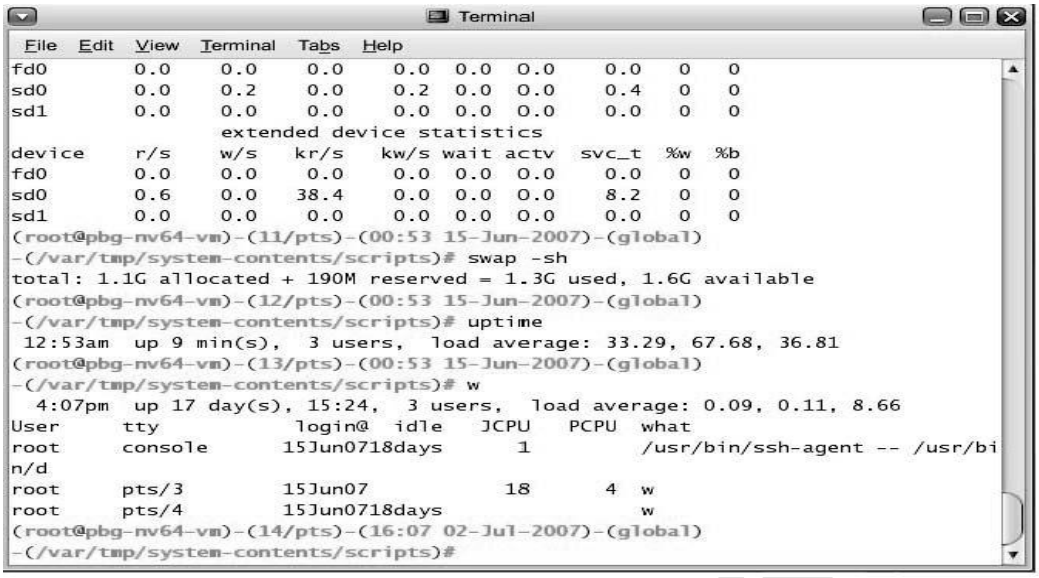
- If a system is to be protected and secure, precautions must be instituted throughout it. A chain is only as strong as its weakest link.

User Operating System Interface - GUI

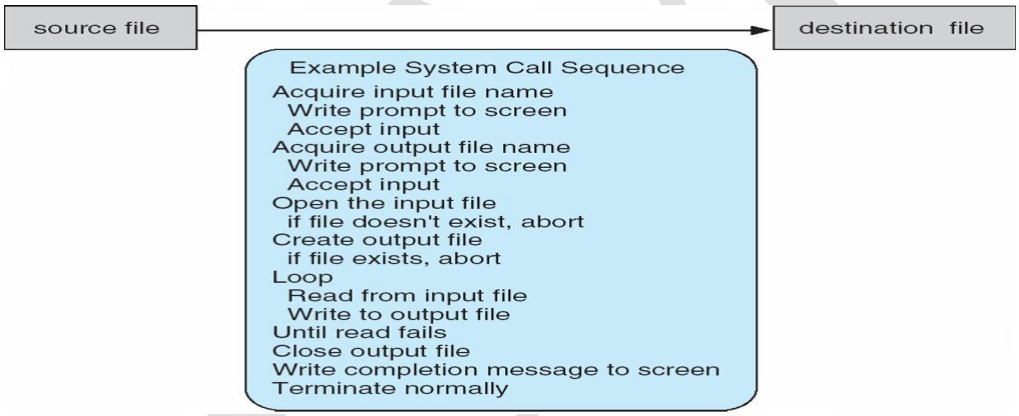
- User-friendly desktop
- metaphor interface Usually
- mouse, keyboard, and monitor Icons represent files, programs, actions, etc

Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a folder)

Bourne Shell Command Interpreter



The Mac OS X GUI



System Calls

- Programming interface to the services
- provided by the OSTypically written in a high-level language (C or C++)
  Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call usenThree most common APIs are Win32 API for Windows, POSIX API for POSIX-basedsystems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)
- Why use APIs rather than system calls?(Note that the system-call names used throughout this text aregeneric)

UNIX

- UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring.

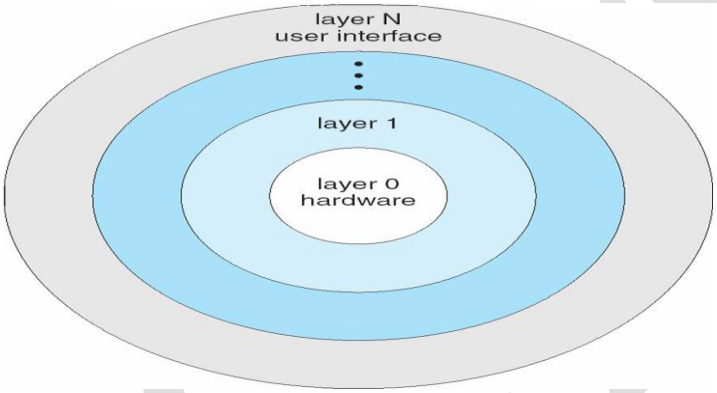
The UNIX OS consists of two

- separable parts Systems
- programs

The kernel

- Consists of everything below the system-call interface and above
- the physical hardware Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level

Layered Operating System



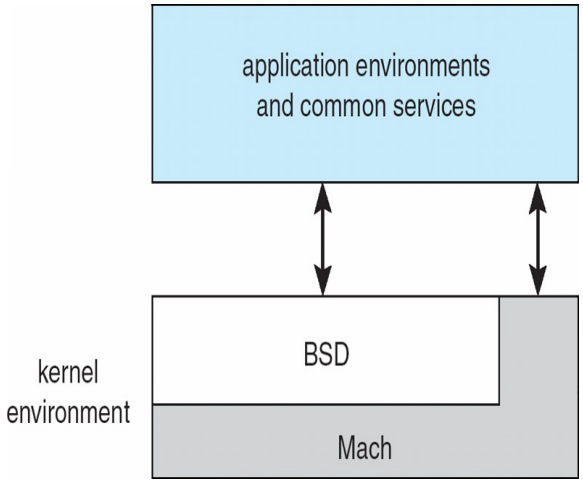
Micro kernel System Structure

- Moves as much from the kernel into "*user*" space
- Communication takes place between user modules using message passing
- Benefits:
- Easier to extend a microkernel
- Easier to port the operating system to new architectures
- More reliable (less code is running in kernel mode)
-

- More
- secure
- Detri ments:

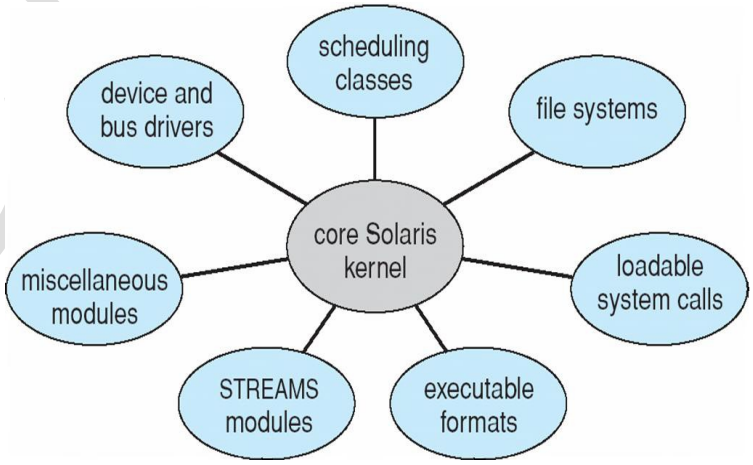
Performance overhead of user space to kernel space communication

Mac OS X Structure



Modules

- Most modern operating systems
- implement kernel modules Uses object-
- oriented approach
- Each core component is separate
- Each talks to the others over known interfaces Each is loadable as needed within the kernel Overall, similar to layers but with more flexible



Solaris Modular Approach

Virtual Machines

- A virtual machine takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware
- A virtual machine provides an interface *identical* to the underlying bare
- hardware
- The operating system host creates the illusion that a process has its own processor and (virtual memory) Each guest provided with a (virtual) copy of underlying computer

UNIT -2

PROCESS MANAGEMENT

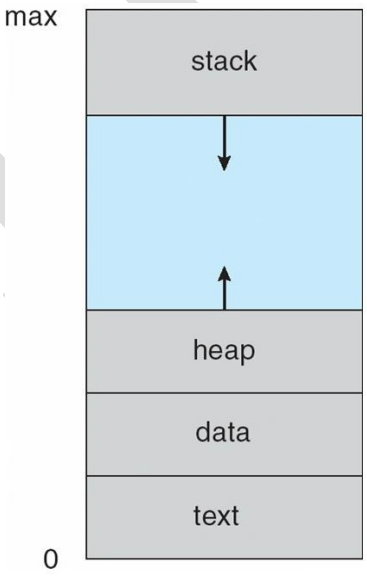
Process Concept

- An operating system executes a variety of programs:
- Batch system – jobs
- Time-shared systems – user programs or tasks
  Textbook uses the terms *job* and *process* almost interchangeably

Process – a program in execution; process execution must progress in sequential fashion A process includes:

- program
- counter
  stack
  data section

Process in Memory

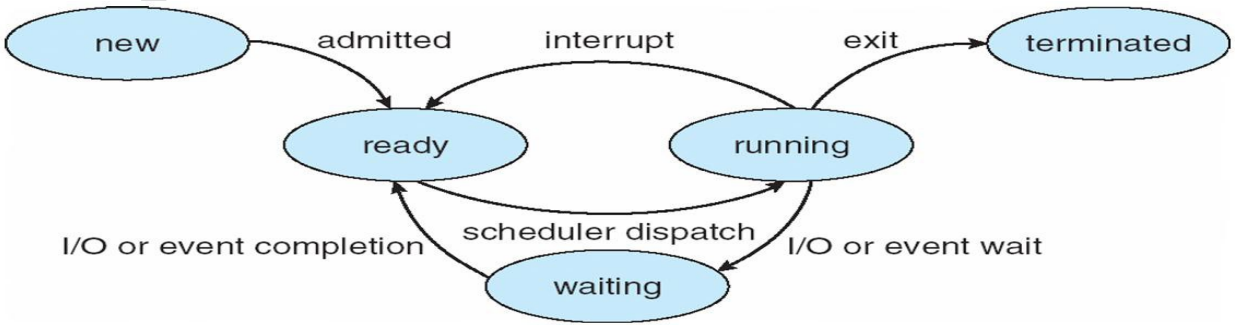


Process State

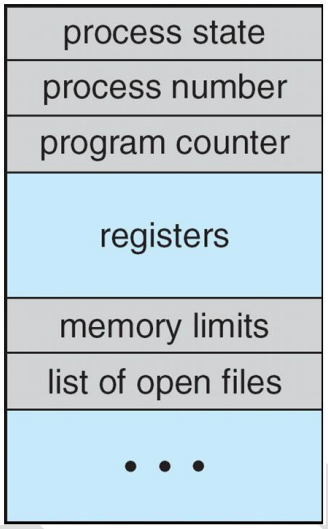
As a process executes, it changes *stat*e

- **new**: The process is being created
- **running**: Instructions are being executed
- **waiting**: The process is waiting for some
- event to occur **ready**: The process is waiting to be assigned to a processor
  terminated: The process has finished execution

Diagram of Process State



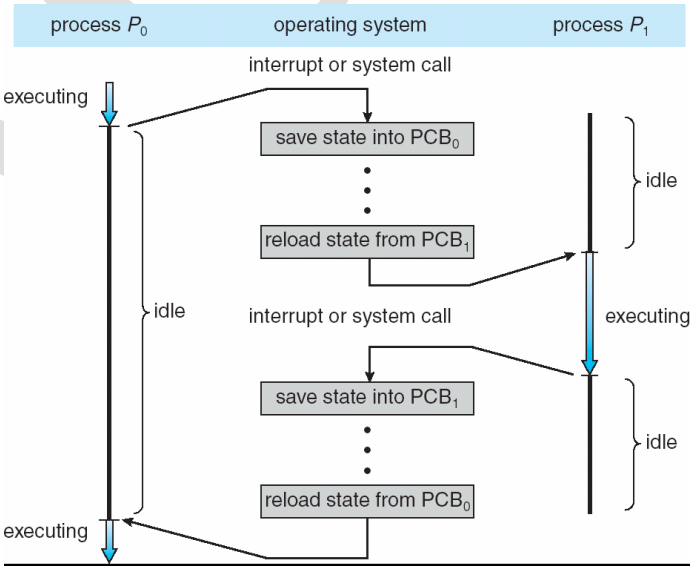
Process Control Block (PCB)



Information associated with

- each process Process
- state
- Program
- counter
- CPU registers
  CPU scheduling information Memory-management information
  Accounting information
  I/O status information

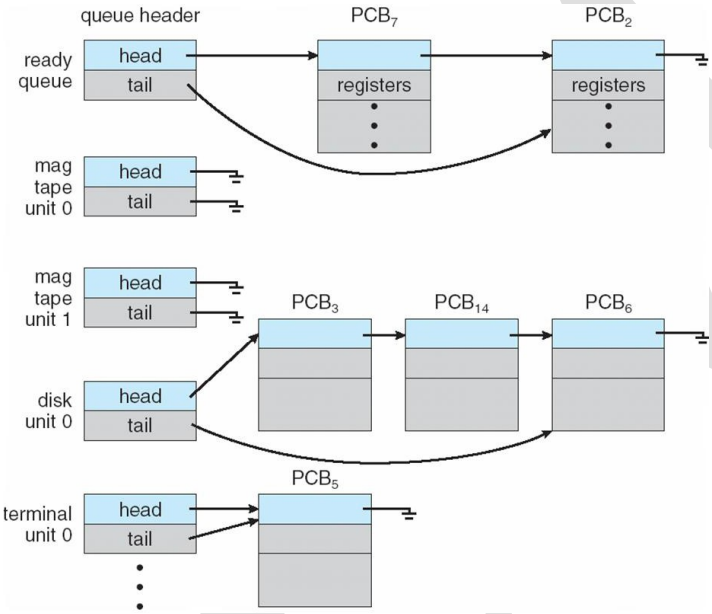
CPU Switch From Process to Process



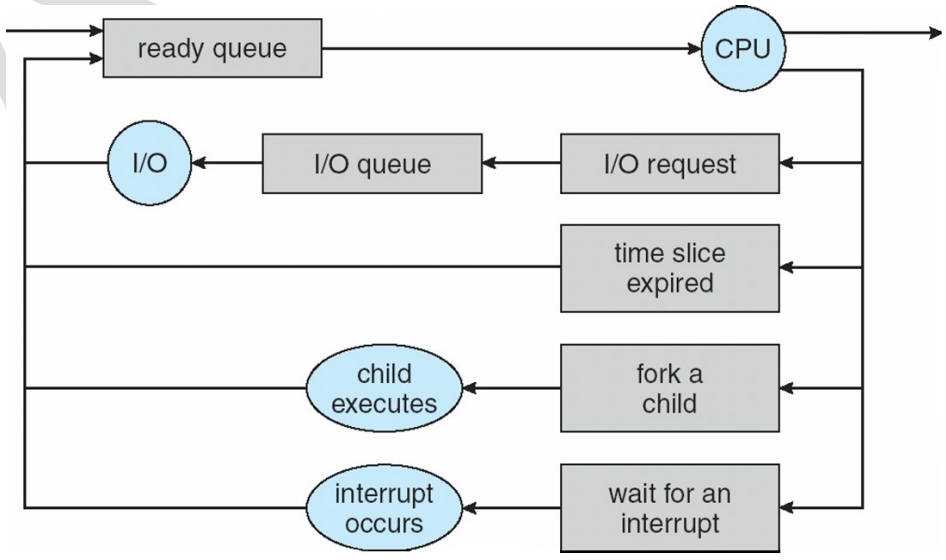
Process Scheduling Queues

- **Job queue** – set of all processes in the system
- **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
- **Device queues** – set of processes waiting for an I/O device Processes migrate among the various queues

Ready Queue and Various I/O Device Queues



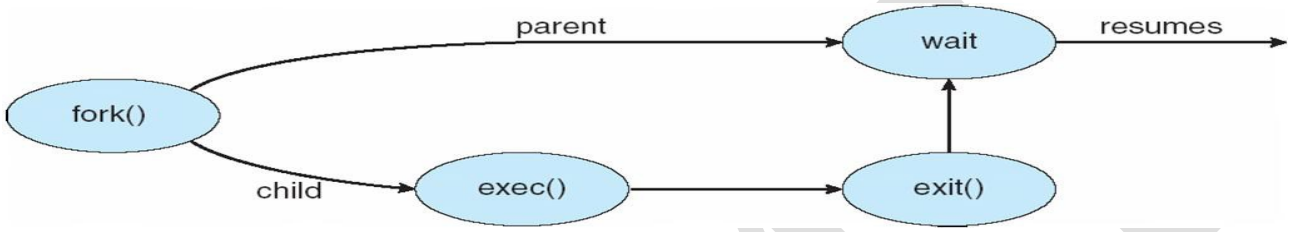
Representation of Process Scheduling



Schedulers

- **Long-term scheduler** (or job scheduler) – selects which processes should be brought into the ready queue
- **Short-term scheduler** (or CPU scheduler) – selects which process should be executed next and allocates CPU

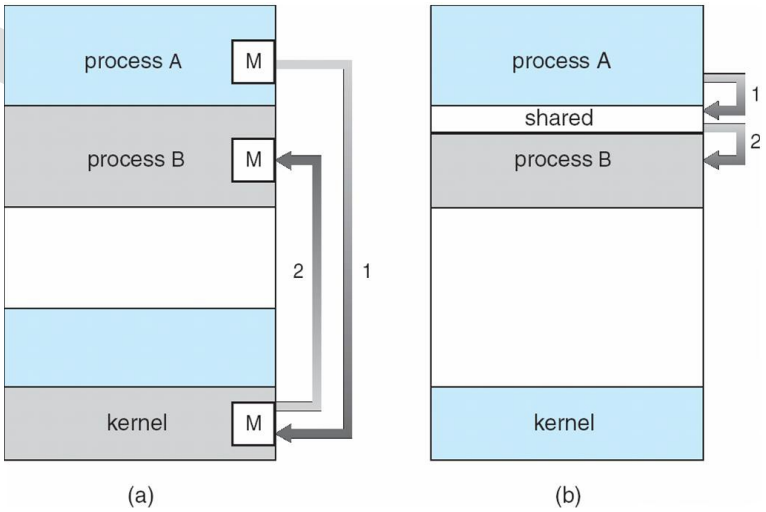
Process Creation



Interprocess Communication

- Processes within a system may be **independent** or **cooperating**
- Cooperating process can affect or be affected by other processes, including sharing data Reasons for cooperating processes:
- Information sharing
- Computation speedup

Communications Models



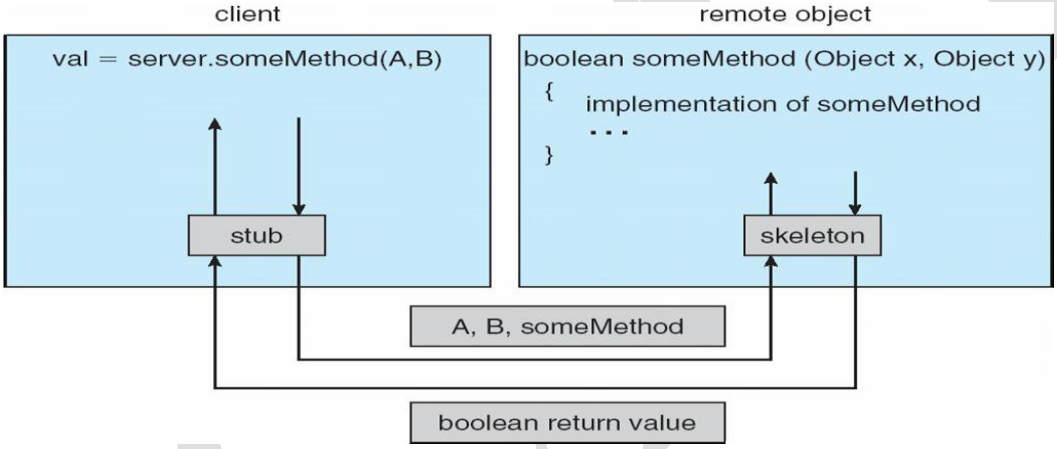
Cooperating Processes

- **Independent** process cannot affect or be affected by the execution of
- another process

Cooperating process can affect or be affected by the execution of another process Advantages of process cooperation

- Information
- sharing
- Computation
- speed-up
- Modularity

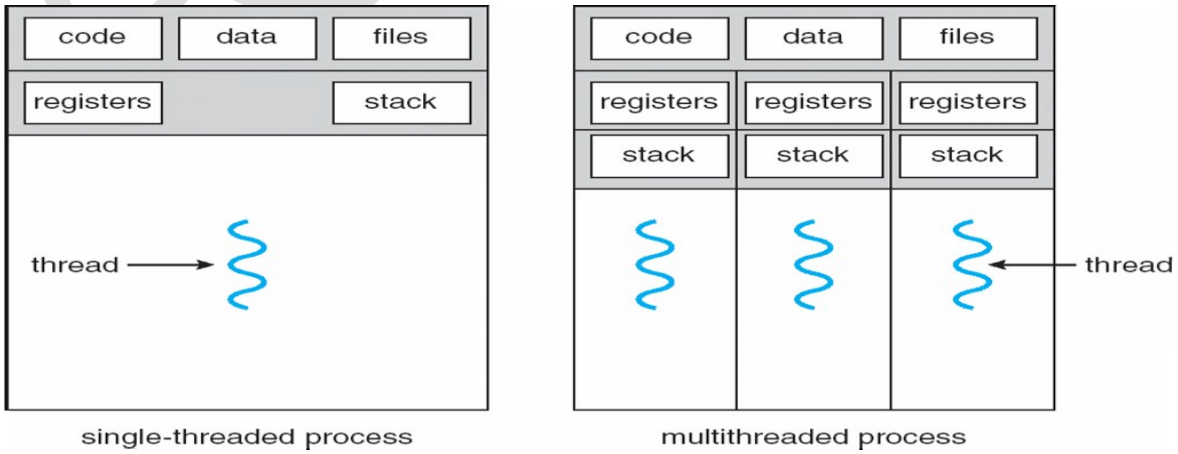
Convenience

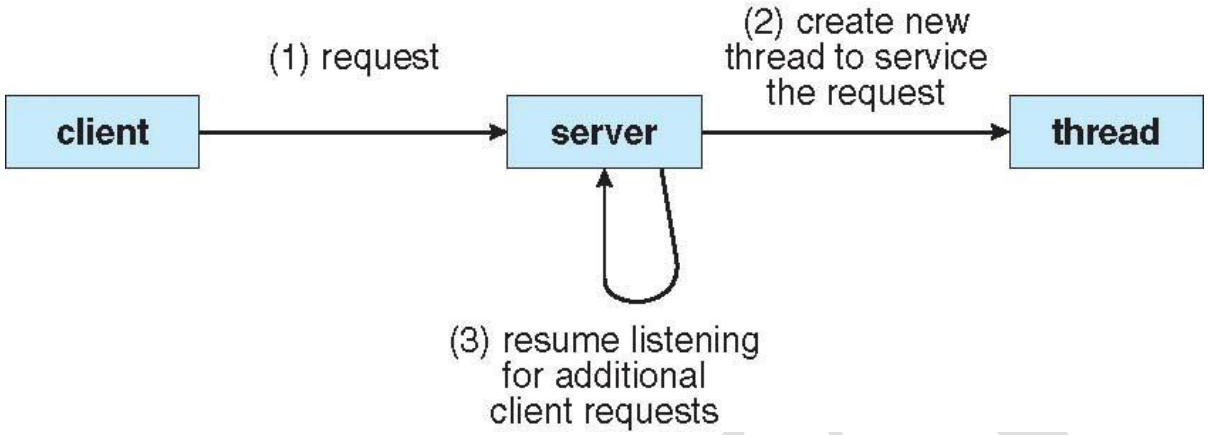


Threads

- To introduce the notion of a thread — a fundamental unit of CPU utilization that forms the basis ofmultithreaded computer systems
- To discuss the APIs for the Pthreads, Win32, and

Single and Multithreaded Processes





Linux Threads

flag	meaning
CLONE_FS	File-system information is shared.
CLONE_VM	The same memory space is shared.
CLONE_SIGHAND	Signal handlers are shared.
CLONE_FILES	The set of open files is shared.

- Linux refers to them as *tasks* rather than *threads*
- Thread creation is done through **clone()** system call
- **clone()** allows a child task to share the address space of the parent task (process)

CPU Scheduling

- To introduce CPU scheduling, which is the basis for
- multiprogrammed operating systemsTo describe various CPU-
- scheduling algorithms
- To discuss evaluation criteria for selecting a CPU-scheduling
- algorithm for a particular systemMaximum CPU utilization obtained with multiprogramming
  CPU–I/O Burst Cycle – Process execution consists of a *cycle* of CPU execution and I/O wait
  CPU burst distribution

CPU Scheduler

Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of themCPU scheduling decisions may take place when a process:

1. Switches from running to waiting state
2. Switches from running to ready state
3. Switches from waiting to ready
4. Terminates

Scheduling under 1 and 4 is **nonpreemptive**All other scheduling is **preemptive**

Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – # of processes that complete their execution per time unit
- **Turnaround time** – amount of time to execute a particular process
- **Waiting time** – amount of time a process has been waiting in the ready queue
- **Response time** – amount of time it takes from when a request was submitted until the first response isproduced, not output (for time-sharing environment)
- Max CPU utilization

First-Come, First-Served (FCFS) Scheduling

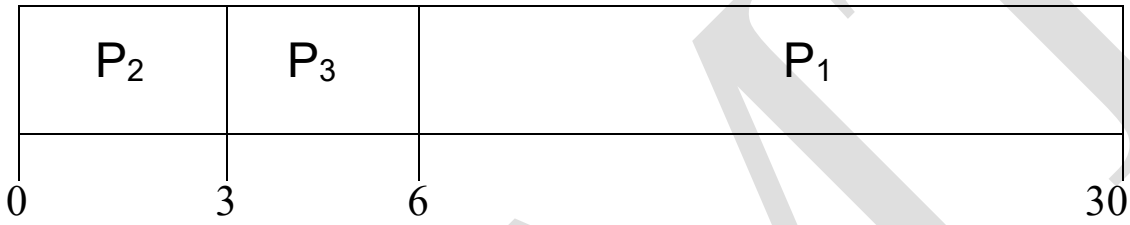
Process	Burst Time
P1	24
P2	3
P3	3

Suppose that the processes arrive in the order: *P1* , *P2* , *P3*
The Gantt Chart for the schedule is:

Waiting time for *P1* = 0; *P2* = 24; *P3* = 27 Average waiting time: (0 + 24 + 27)/3 = 17 Suppose that the processes arrive in the order

P2 , *P3* , *P1*

The Gantt chart for the schedule is:nnnnWaiting time for *P1* = 6*; P2* = 0*; P3* = 3nAverage waiting time: (6 +0 + 3)/3 = 3
Much better than previous case
Convoy effect short process behind long process



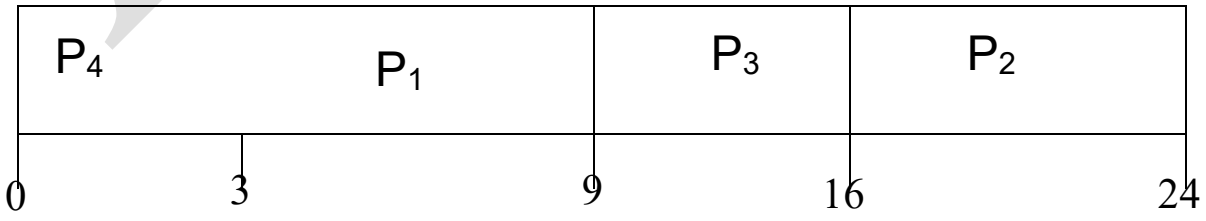
Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the processwith the shortest time
- SJF is optimal – gives minimum average waiting time for

a given set of processesThe difficulty is knowing

Process	Arrival Time	Burst Time
P1	0.0	6
P2	2.0	8
P3	4.0	7
P4	5.0	3

JF scheduling chart
average waiting time = (3 + 16 + 9 + 0) / 4 = 7the length of the next CPU request



Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer º highest priority)
- Preempt
- ive

  nonpree

  mptive

  SJF is a priority scheduling where priority is the predicted next CPU burst timeProblem º **Starvation** – low priority processes may never execute

  Solution º **Aging** – as time progresses increase the priority of the process

Round Robin (RR)

- Each process gets a small unit of CPU time (*time quantum*), usually 10-100 milliseconds. After thistime has elapsed, the process is preempted and added to the end of the ready queue.
- If there are *n* processes in the ready queue and the time quantum is *q*, then each process gets 1/*n* of theCPU time in chunks of at most *q* time units at once. No process waits more than (*n*-1)*q* time units.
- Performance

  q large Þ FIFO

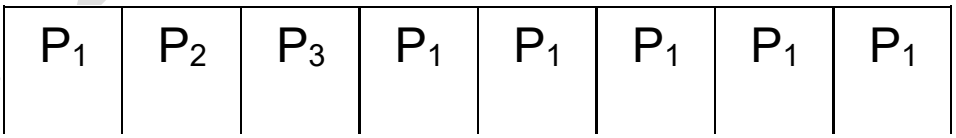
  q small Þ *q* must be large with respect to context switch, otherwise overhead is too high

Example of RR with Time Quantum = 4

Process	Burst Time
P1	24
P2	3
P3	3

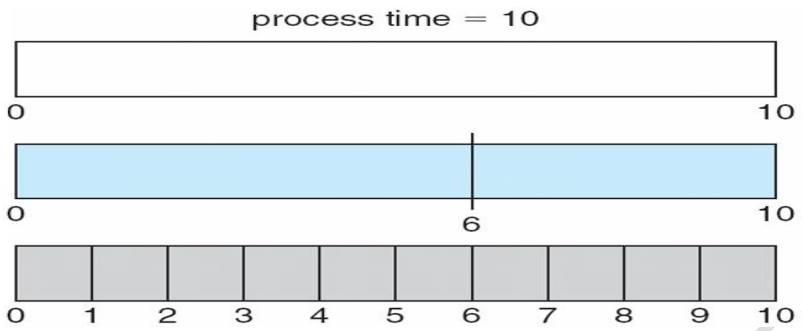
The Gantt chart is:

Typically,



0 4 7 10 14 18 22 26 30

Time Quantum and Context Switch Time



Thread Scheduling

- Distinction between user-level and kernel-level threads
- Many-to-one and many-to-many models, thread library schedules user-level threads to run on LWP Known as **process-contention scope (PCS)** since scheduling competition is within the process Kernel thread scheduled onto available CPU is **system-contention scope (SCS)** – competition among all threads in system

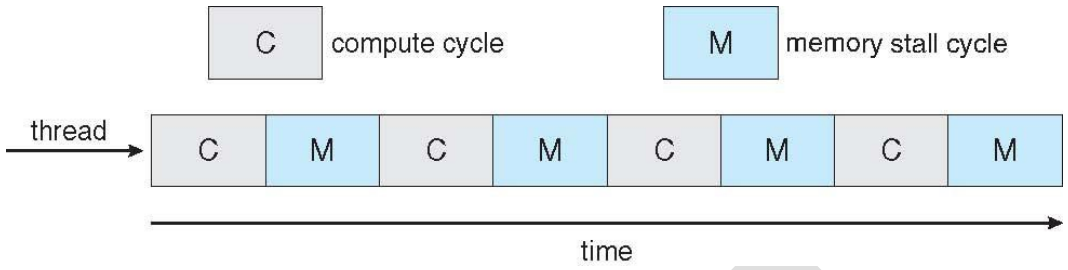
Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available
- **Homogeneous processors** within a multiprocessor
  Asymmetric multiprocessing – only one processor accesses the system data structures, alleviating the need for data sharing
- **Symmetric multiprocessing (SMP)** – each processor is self-scheduling, all processes in common ready queue, or each has its own private queue of ready processes
- **Processor affinity** – process has affinity for processor on which it is currently running

Multicore Processors

- Recent trend to place multiple processor cores on same physical chip Faster and consume less power
  Multiple threads per core also growing
  Takes advantage of memory stall to make progress on another thread while memory retrieve happens

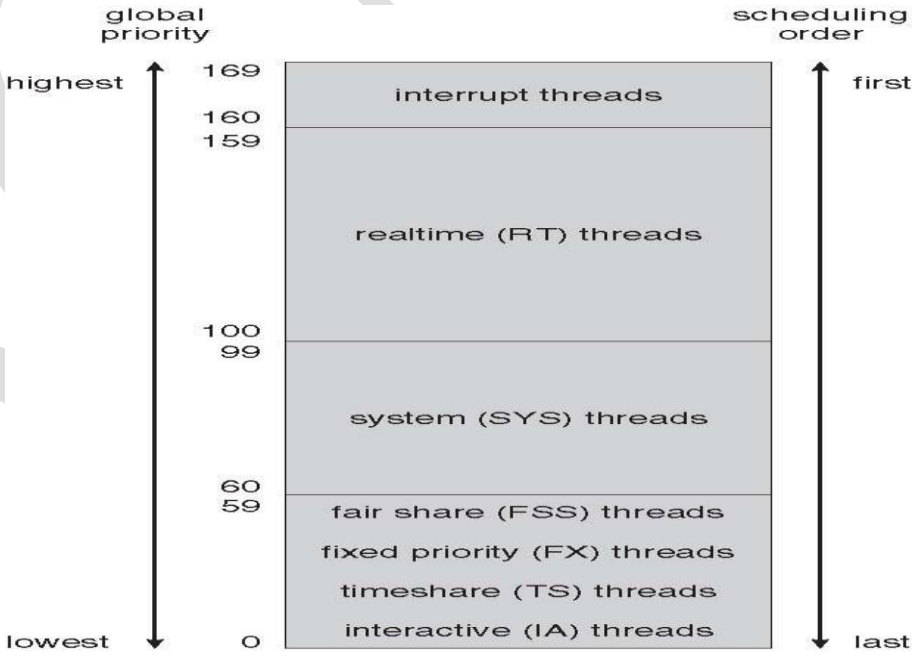
Multithreaded Multicore System



Operating System

- **Examples**
- Solaris scheduling
  Windows XP scheduling
  Linux scheduling

Solaris Scheduling



Windows XP Priorities

	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

Linux Scheduling

- Constant order *O*(1) scheduling time
- Two priority ranges: time-sharing and real-time
- **Real-time** range from 0 to 99 and **nice** value from 100 to 140

UNIT -3
CONCURRENCY

Process Synchronization

- To introduce the critical-section problem, whose solutions can be used to ensure the consistency ofshared data
- To present both software and hardware solutions of the critical-section problem
- To introduce the concept of an atomic transaction and describe mechanisms to ensure atomicityConcurrent access to shared data may result in data inconsistency
  Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperatingprocesses
- Suppose that we wanted to provide a solution to the consumer-producer problem that fills all the buffers. We can do so by having an integer count that keeps track of the number of full buffers. Initially, count is set to 0. It is incremented by the producer after it produces a new buffer and is decremented by the consumer after it consumes a buffer

Peterson's Solution

- Two process solution
- Assume that the LOAD and STORE instructions are atomic; that
- is, cannot be interrupted. The two processes share two variables:
- int turn;
- Boolean
  flag[2]
  section

Synchronization Hardware

- Many systems provide hardware support for
- critical section codeUniprocessors – could
- disable interrupts
  Currently running code would execute
  without preemptionGenerally too
  inefficient on multiprocessor systems
  Operating systems using this not broadly scalable
- Modern machines provide special atomic

Semaphore

- Synchronization tool that does not require busy waiting
- nSemaphore *S* – integer variableTwo standard operations
- modify S: wait() and signal()

Semaphore Implementation

- Must guarantee that no two processes can execute wait () and signal () on the same semaphore at thesame time
- Thus, implementation becomes the critical section problem where the wait and signal code are placed inthe crtical section.
- Could now have busy waiting in critical

section implementationBut implementation code
is short
Little busy waiting if critical section rarely occupied

- Note that applications may spend lots of time in critical sections and therefore this is not a goodsolution.

Deadlock and Starvation

- Deadlock – two or more processes are waiting indefinitely for an event that can be caused by only oneof the waiting processes
- Let S and Q be two semaphores initialized to 1

Monitors

A high-level abstraction that provides a convenient and effective mechanism for process synchronizationOnly one process may be active within the monitor at a time
monitor monitor-name

- spin locks

System Model

- Assures that operations happen as a single logical unit of work,
- in its entirety, or not at allRelated to field of database systems
- Challenge is assuring atomicity despite computer system failures
- Transaction - collection of instructions or operations that
- performs single logical functionHere we are concerned with changes to stable storage – disk

Transaction is series of read and write operations
Terminated by commit (transaction successful) or abort (transaction failed) operation Abortedtransaction must be rolled back to undo any changes it performed

Types of Storage Media

- Volatile storage – information stored here does not
- survive system crashesExample: main memory,
- cache
- Nonvolatile storage – Information
- usually survives crashesExample: disk and tape
  Stable storage – Information never lost
  Not actually possible, so approximated via replication or RAID to devices with independent failuremodes
- Goal is to assure transaction atomicity where failures cause loss of information on volatile storage

Concurrent Transactions

- Must be equivalent to serial execution
- – serializabilityCould perform all
- transactions in critical section
  Inefficient, too restrictive
  Concurrency-control algorithms provide serializability

Serializability

- Consider two data items
- A and B Consider
- Transactions T0 and T1
- Execute T0, T1
- atomically Execution sequence called schedule

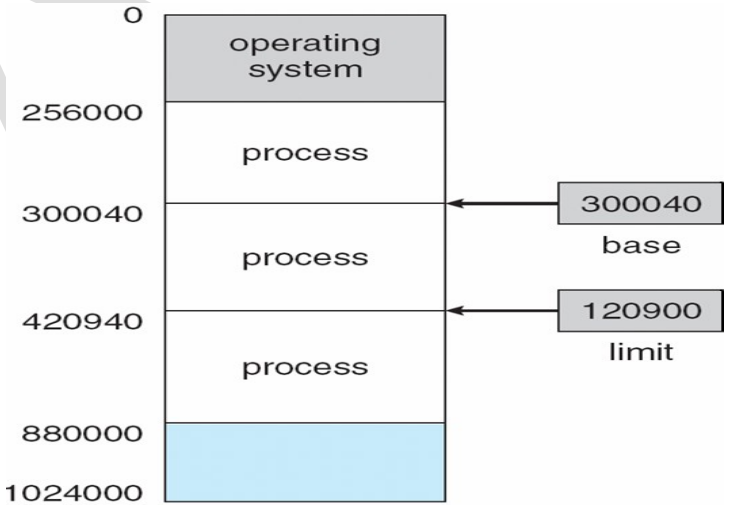
UNIT IV

Memory Management

- To provide a detailed description of various ways of organizing memory
- hardware
- To discuss various memory-management techniques, including paging and segmentation
  To provide a detailed description of the Intel Pentium, which supports both pure segmentation andsegmentation with paging
- Program must be brought (from disk) into memory and placed
- within a process for it to be runMain memory and registers are only
- storage CPU can access directly
- Register access in one CPU clock (or less)Main memory can take many cycles
  Cache sits between main memory and CPU registers Protection of memory required to ensure correct operation

Base and Limit Registers

A pair of **base** and **limit** registers define the logical address space

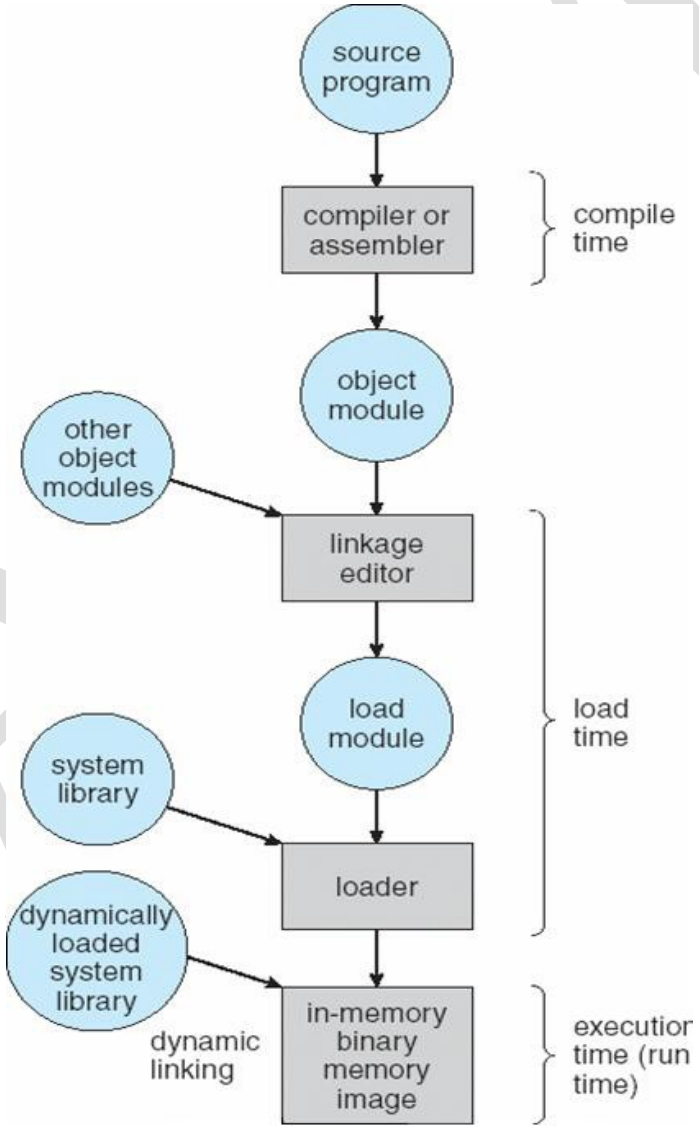


Binding of Instructions and Data to Memory

- Address binding of instructions and data to memory addresses can
- happen at three different stages **Compile time**: If memory location
- known a priori, **absolute code** can be generated; must recompilecode if
- starting location changes

  Load time: Must generate **relocatable code** if memory location is not known at compile time **Execution time**: Binding delayed until run time if the process can be moved during its execution fromone memory segment to another. Need hardware support for address maps (e.g., base and limit registers)

Multistep Processing of a User Program



Logical vs. Physical Address Space

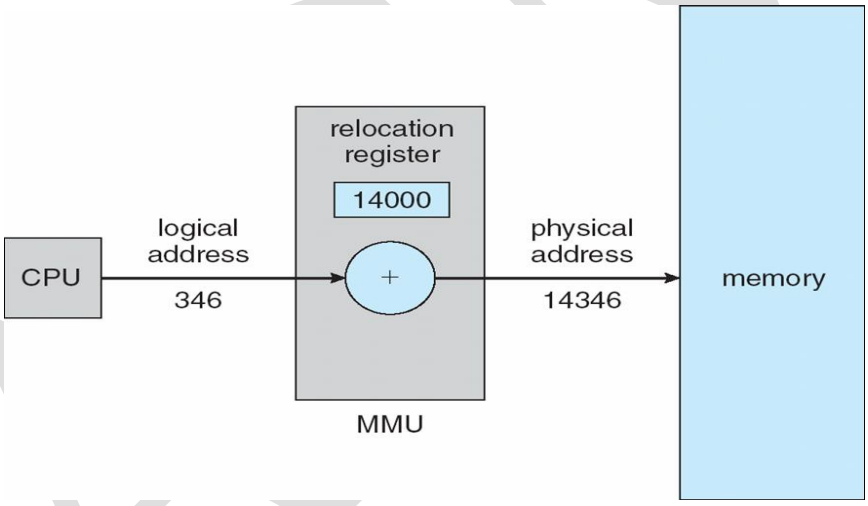
- The concept of a logical address space that is bound to a separate **physical address space** is central to proper memory management
- **Logical address** – generated by the CPU; also referred
- to as **virtual address** **Physical address** – address seen by the memory unit

  Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme

Memory-Management Unit (MMU)

- Hardware device that maps virtual to physical address
- In MMU scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory
- The user program deals with *logical* addresses; it never sees the *real* physical addresses

Dynamic relocation using a relocation register



Dynamic Loading

- Routine is not loaded until it is called
- Better memory-space utilization; unused routine is never loaded
- Useful when large amounts of code are needed to handle infrequently occurring cases

  No special support from the operating system is required implemented through program design

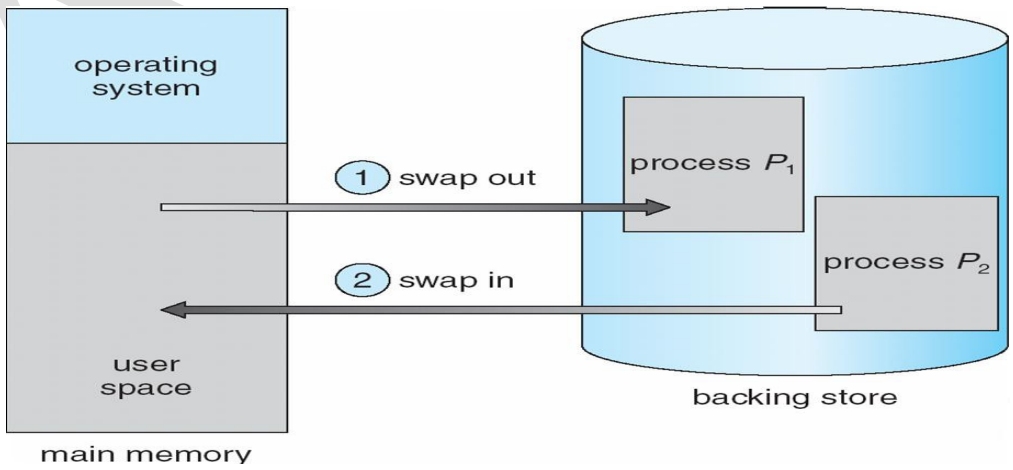
Dynamic Linking

- Linking postponed until execution time
- Small piece of code, *stub*, used to locate the appropriate
- memory-resident library routineStub replaces itself with the
- address of the routine, and executes the routine
- Operating system needed to check if routine is in processes' memory addressDynamic linking is particularly useful for libraries
  System also known as **shared libraries**

Swapping

A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued executionn**Backing store** – fast disk large enough to accommodate copies of all memory imagesfor all users; must provide direct access to these memory imagesn**Roll out, roll in** – swapping variant used forpriority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executednMajor part of swap time is transfer time; total transfer time is directly proportional to theamount of memory swappednModified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)
System maintains a **ready queue** of ready-to-run processes which have memory images on disk

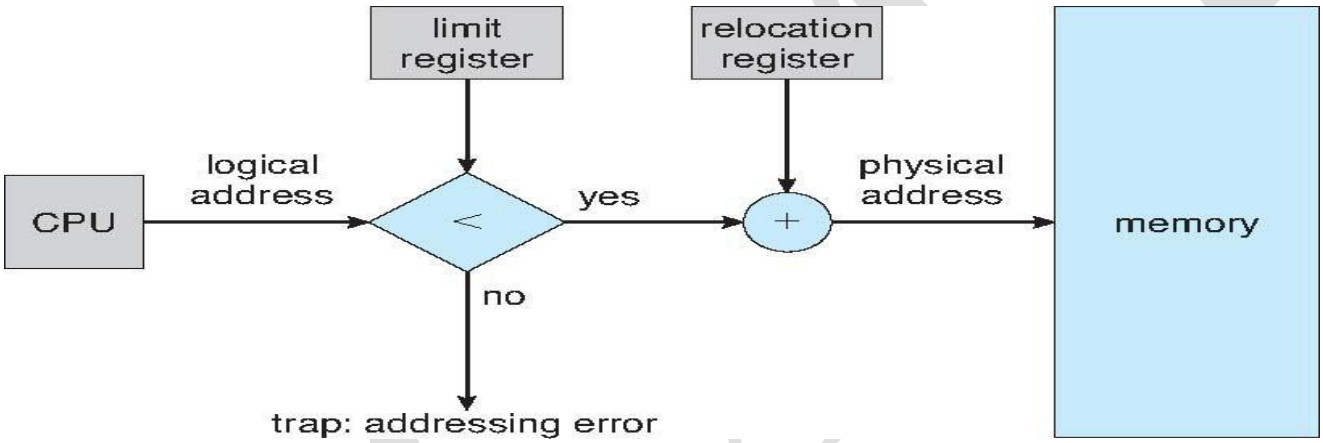
Schematic View of Swapping



Contiguous Allocation

- Main memory usually into two partitions:
- Resident operating system, usually held in low memory with interrupt vector
- User processes then held in high memorynRelocation registers used to protect user processes from eachother, and from changing operating-
- system code and data
  Base register contains value of smallest physical address
  Limit register contains range of logical addresses – each logical address must be less than the limitregister
- MMU maps logical address *dynamically*

Hardware Support for Relocation and Limit Registers



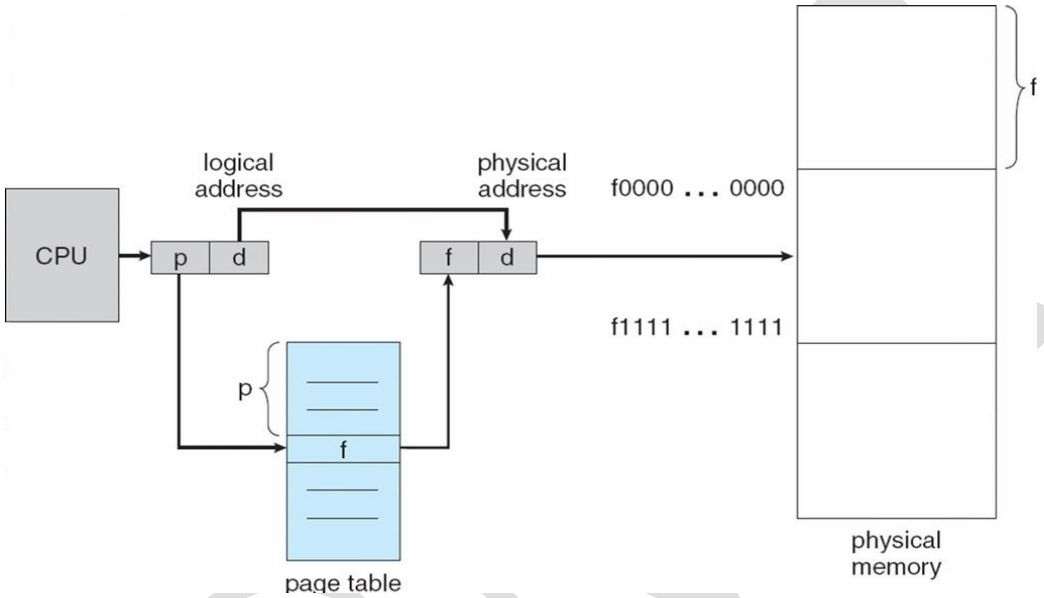
- To run a program of size ***n*** pages, need to find *n* free
- frames and load programSet up a page table to translate
- logical to physical addresses
  Internal fragmentation

Address Translation Scheme

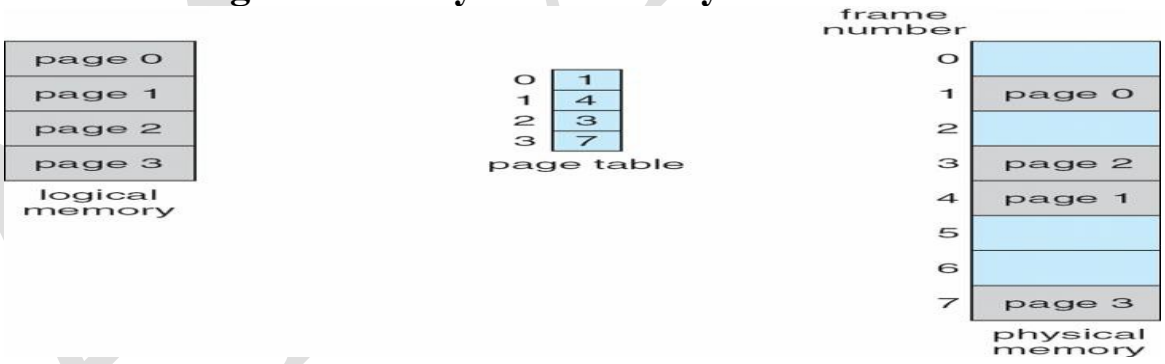
- Address generated by CPU is divided into
- **Page number (*p*)** – used as an index into a *page table* which contains base address of each page inphysical memory

- **Page offset (d)** – combined with base address to define the physical memory address that is sent to thememory unit
- For given logical address space 2^m *and page size* 2^n

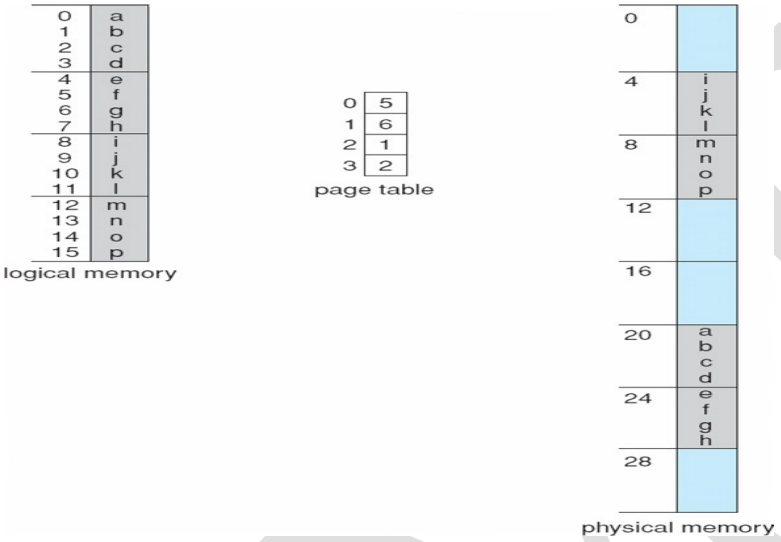
Paging Hardware



Paging Model of Logical and Physical Memory

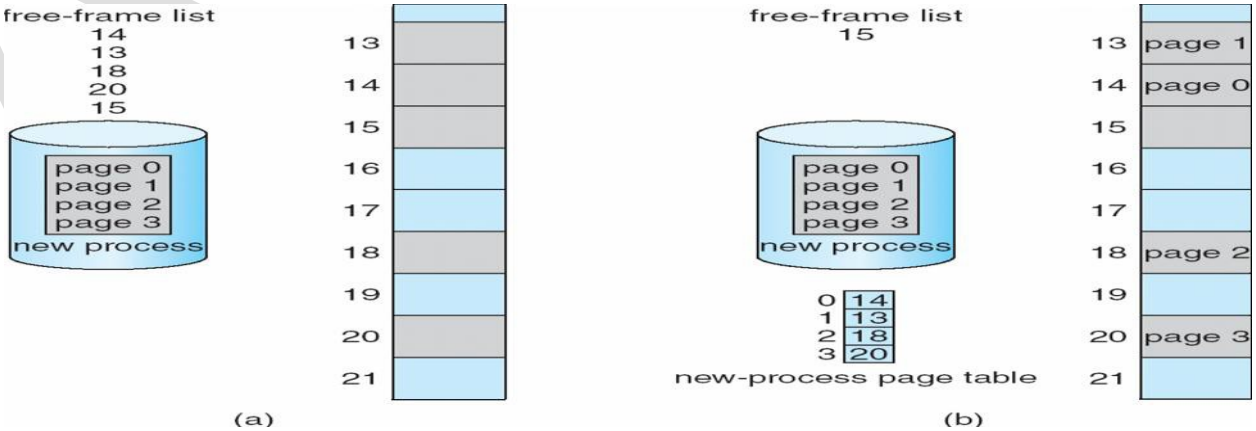


Paging Example



32-byte memory and 4-byte pages

Free Frames



Implementation of Page Table

- Page table is kept in main memory
- **Page-table base register (PTBR)** points to the page table
- **Page-table length register (PRLR)** indicates size of the page table
  In this scheme every data/instruction access requires two memory accesses. One for the page table andone for the data/instruction.
- The two memory access problem can be solved by the use of a special fast-lookup hardware cachecalled **associative memory** or **translation look-aside buffers (TLBs)**
- Some TLBs store **address-space identifiers (ASIDs)** in each TLB entry – uniquely identifies eachprocess to provide address-space protection for that process

Associative Memory

- Associative memory –
- parallel searchAddress
- translation (p, d)
  If p is in associative register, get frame # out Otherwise get frame # from page table in memory

Page #	Frame #

Effective Access Time

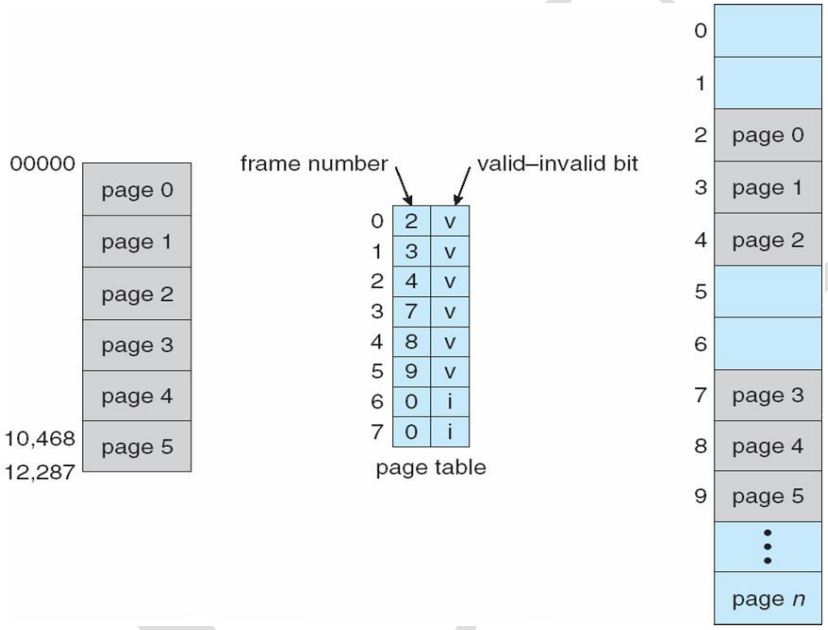
- Associative Lookup = e time unit
- Assume memory cycle time is 1 microsecond
- Hit ratio – percentage of times that a page number is found in the associative registers; ratio related tonumber of associative registers
- Hit ratio = an **Effective Access Time** (EAT)

$$EAT = (1 + e)\, a + (2 + e)(1 - a)$$
$$= 2 + e - a$$

Memory Protection

- Memory protection implemented by associating protection bit with each
- frame
- **Valid-invalid** bit attached to each entry in the page table:
- "valid" indicates that the associated page is in the process' logical address space, and is thus a legal page"invalid" indicates that the page is not in the process' logical address space

Valid (v) or Invalid (i) Bit In A Page Table



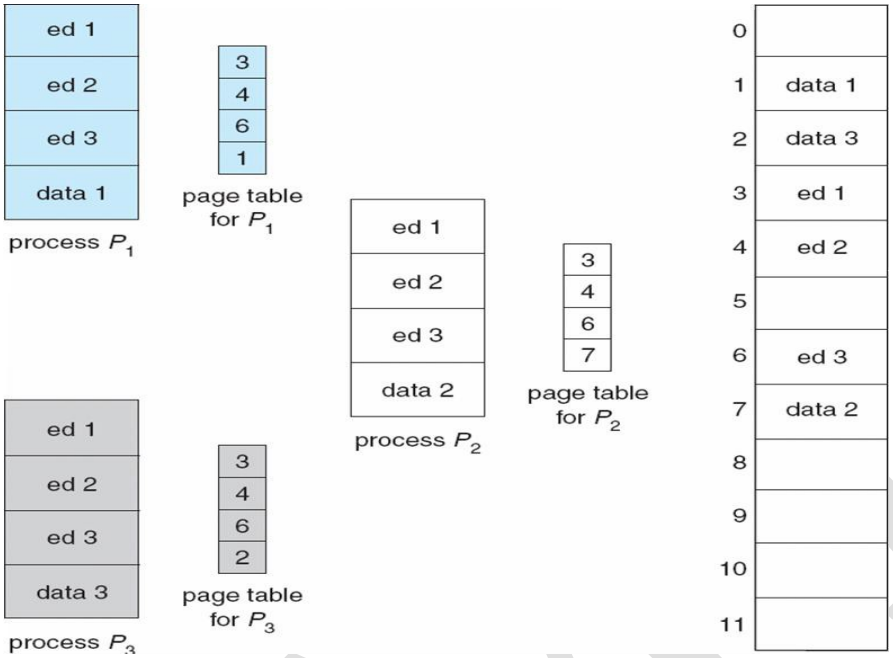
Shared PagesShared code

- One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, windowsystems).
- Shared code must appear in same location in the logical address space of all processes

Private code and data

- Each process keeps a separate copy of the code and data
- The pages for the private code and data can appear anywhere in the logical address space

Shared Pages Example



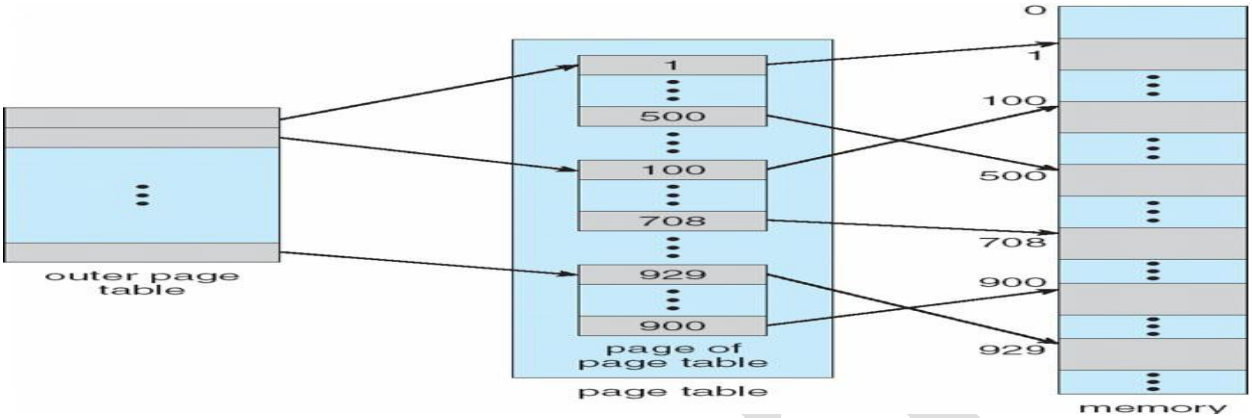
Structure of the Page Table

- Hierarchical
- Paging
- Hashed Page Tables
  Inverted Page Tables

Hierarchical Page Tables

- Break up the logical address space into
- multiple page tables A simple technique is a two-level page table

Two-Level Page-Table Scheme



Two-Level Paging Example

- A logical address (on 32-bit machine with 1K page
- size) is divided into: a page number consisting of
- 22 bits
- a page offset consisting of 10 bits
- Since the page table is paged, the page number is further divided into:
  a 12-bit page
  numbera 10-
  bit page
  offset
  Thus, a logical address is as follows:

  where *pi* is an index into the outer page table, and *p2* is the displacement within the page of the outerpage table

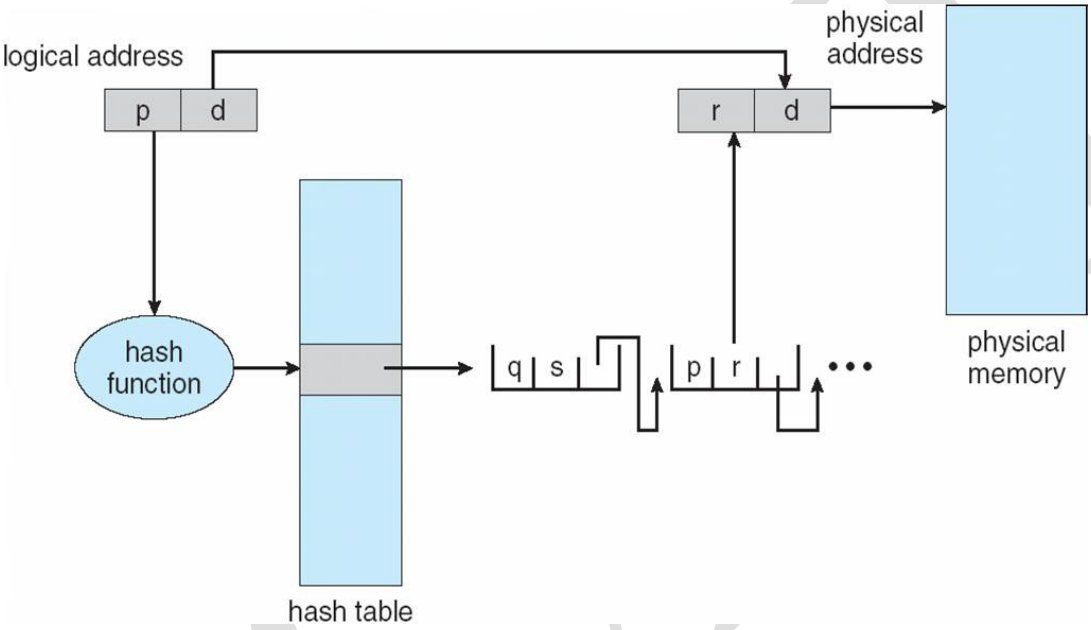
page number		page offset
p_i	p_2	d
12	10	10

Hashed Page Tables

Common in address spaces > 32 bits

The virtual page number is hashed into a page table
This page table contains a chain of elements
hashing to the same location Virtual page numbers
are compared in this chain searching for a match
If a match is found, the corresponding physical frame is extracted

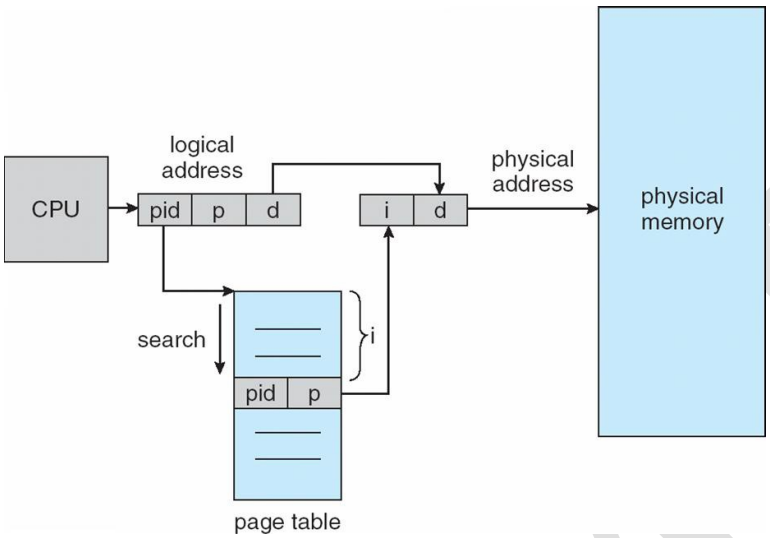
Hashed Page Table



Inverted Page Table

- One entry for each real page of memory
- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page
- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs
- Use hash table to limit the search to one — or at most a few — page-table entries

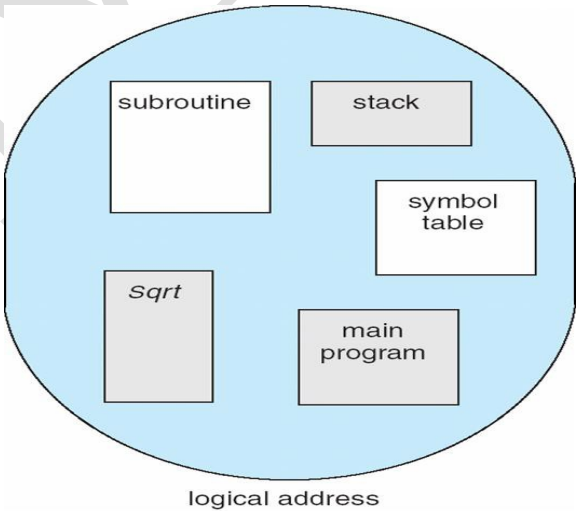
Inverted Page Table Architecture



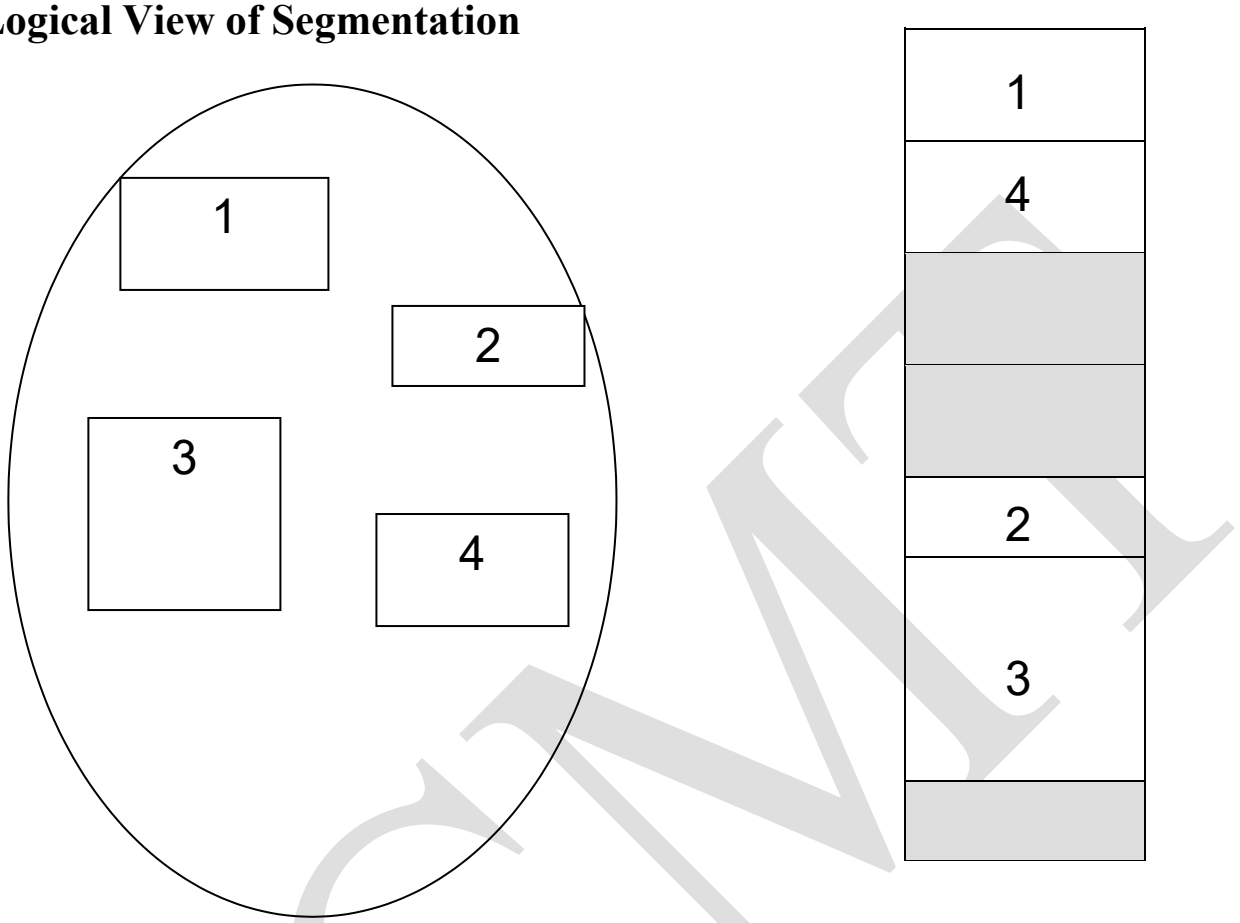
Segmentation

Memory-management scheme that supports

- user view of memory A program is a
- collection of segments
- A segment is a logical
- unit such as: main
- program
- procedure
- function
- method
  object
  local variables,
  global variables
  common block
  stack
  symbo
  l table
  arrays



Logical View of Segmentation



user space

Segmentation Architecture

- Logical address consists of a two tuple:
  - <segment-number, offset>,
- **Segment table** – maps two-dimensional physical addresses; each table entry has:
  - **base** – contains the starting physical address where the segments reside in memory
  - **limit** – specifies the length of the segment

Segment-table base register (STBR) points to the segment table's location in memory

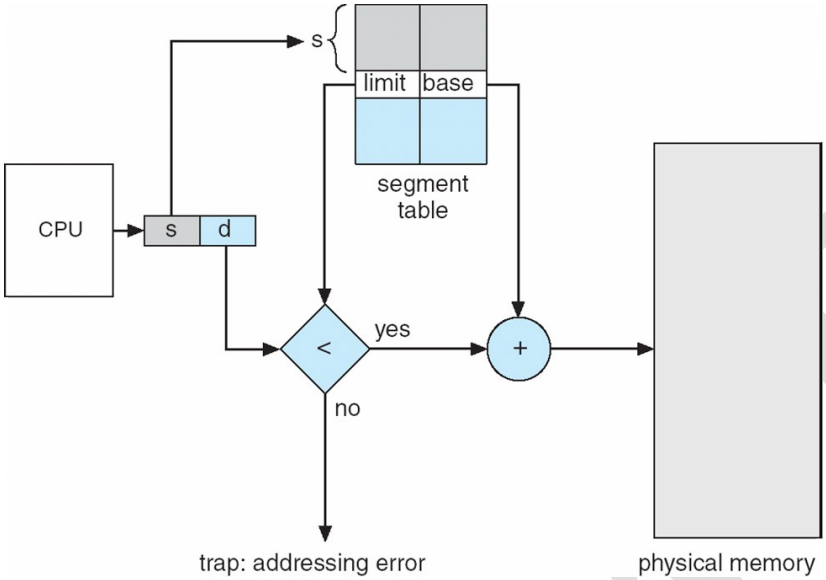
Segment-table length register (STLR) indicates number of segments used by a program; segment number ***s*** is legal if ***s*** < **STLR**

Protection

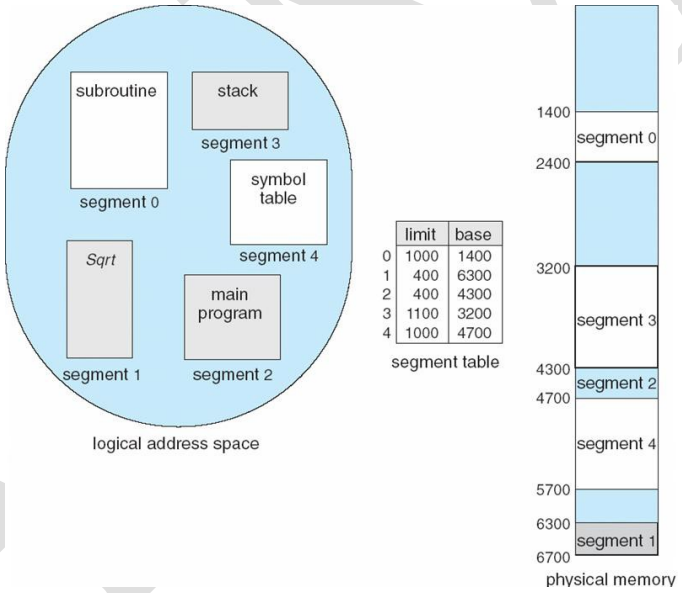
With each entry in segment table associate:

- validation bit = 0 Þ

Segmentation Hardware



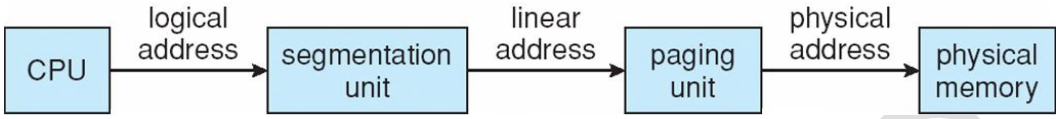
Example of Segmentation

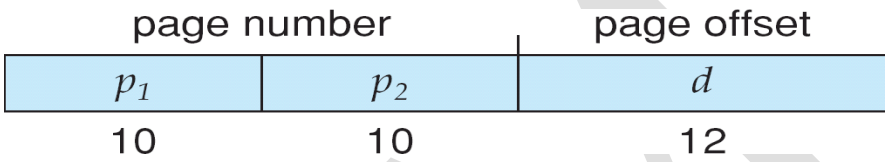


Example: The Intel Pentium

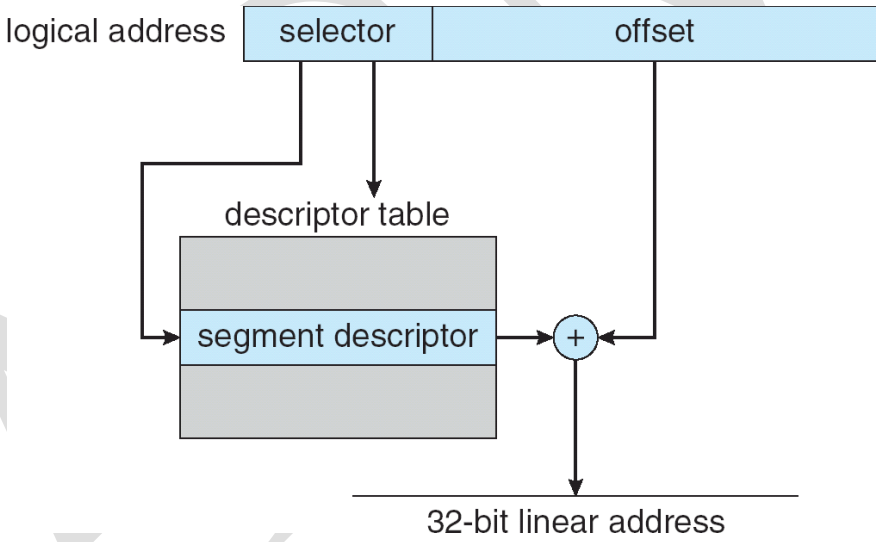
- Supports both segmentation and
- segmentation with pagingCPU generates logical address
  Given to segmentation unit
  - Which produces linear addresses • Linear address given to paging unit

Logical to Physical Address Translation in Pentium

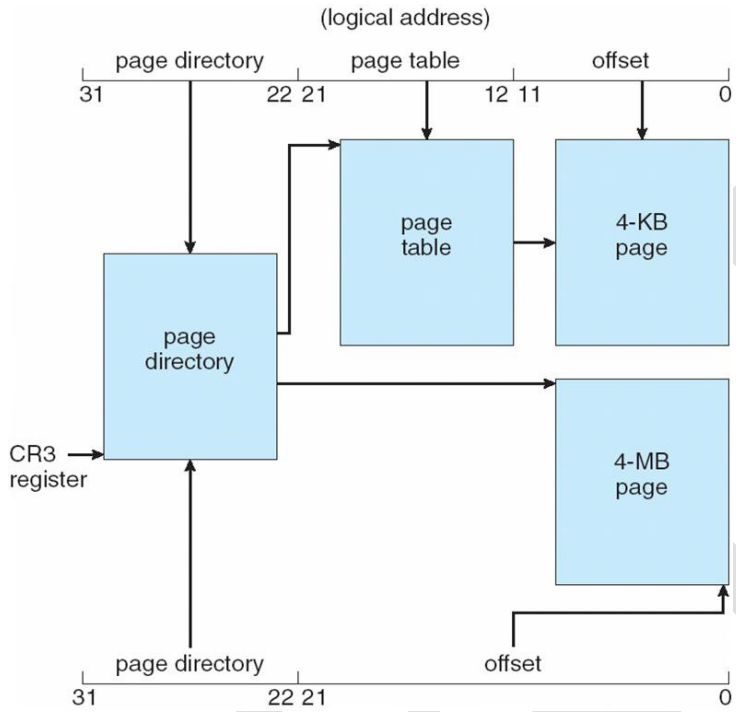




Intel Pentium Segmentation

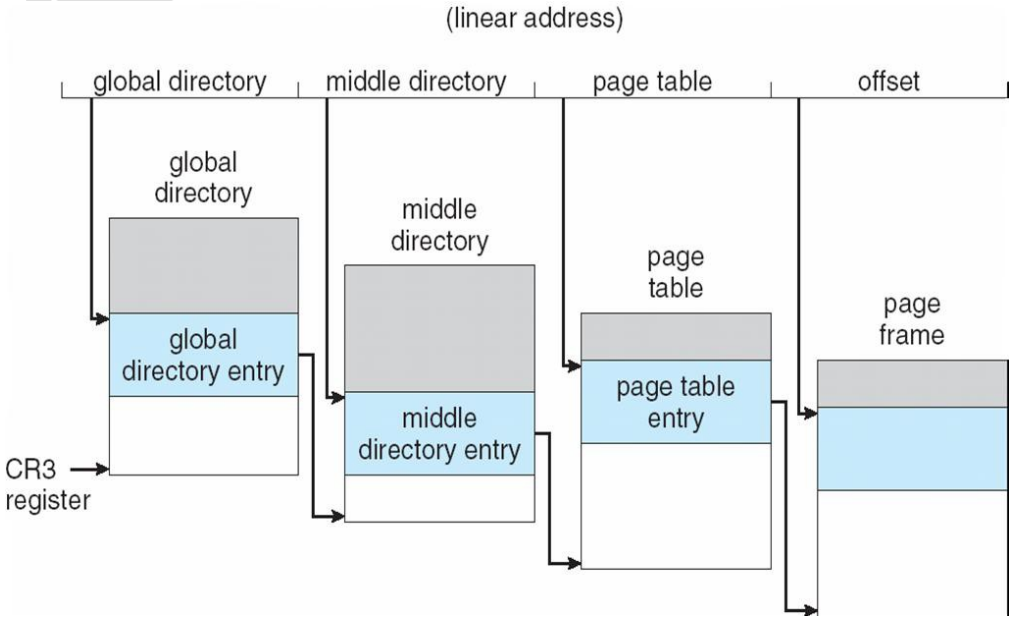


Pentium Paging Architecture

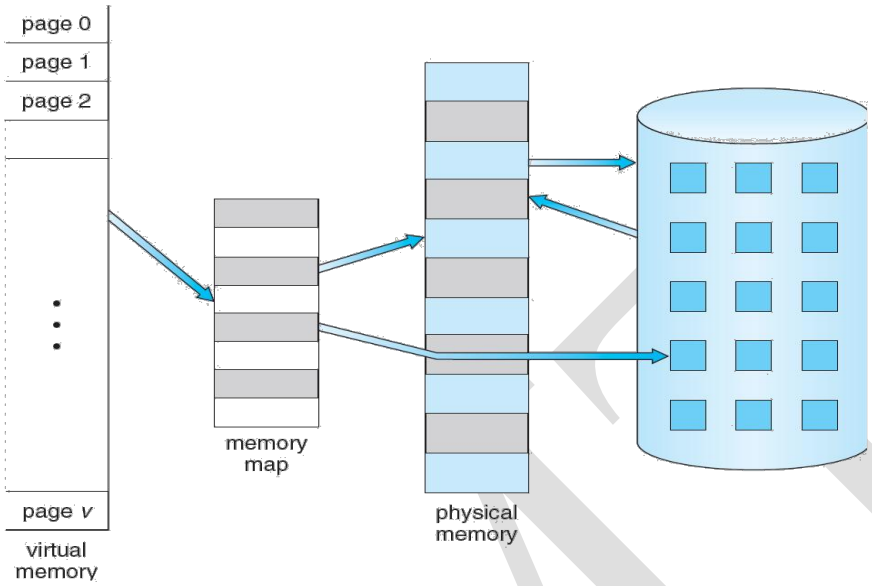


Linear Address in Linux

Three-level Paging in Linux



UNIT – 5



VIRTUAL MEMORY

Objective

- To describe the benefits of a virtual memory system.
- To explain the concepts of demand paging, page-replacement algorithms, and allocation of page frames.
- To discuss the principle of the working-set model.

Virtual Memory

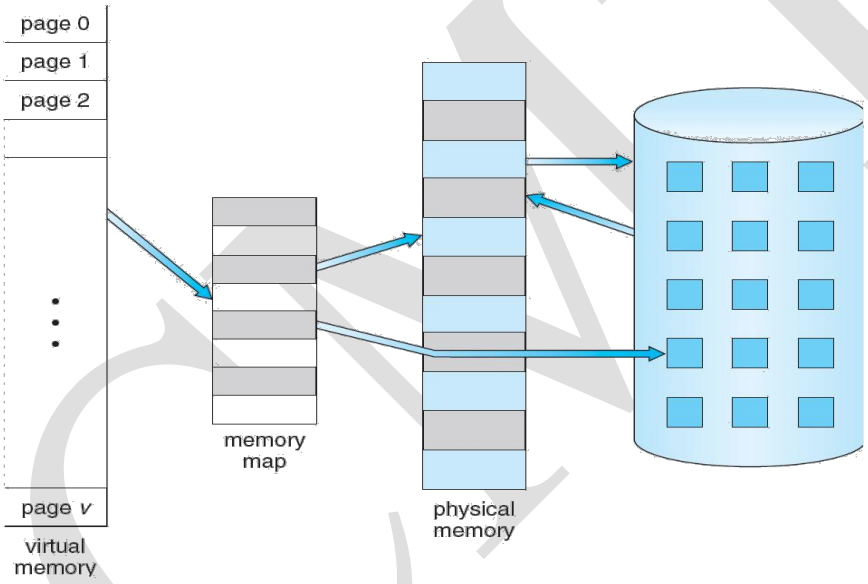
- Virtual memory is a technique that allows the execution of process that may not be completely in memory. The main visible advantage of this scheme is that programs can be larger than physical memory.
- Virtual memory is the separation of user logical memory from physical memory this separation allows an extremely large virtual memory to be provided for programmers when only a smaller physical memory is available (Fig).
- Following are the situations, when entire program is not required to load fully.

1. User written error handling routines are used only when an error occurs in the data or computation.
2. Certain options and features of a program may be used rarely.
3. Many tables are assigned a fixed amount of address space even though

only a small amount of the table is actually used.

■ The ability to execute a program that is only partially in memory would counter many benefits.

1. Less number of I/O would be needed to load or swap each user program into memory.
2. A program would no longer be constrained by the amount of physical memory that is available.
3. Each user program could take less physical memory, more programs could be run the same time, with a corresponding increase in CPU utilization and throughput.



1. We check an internal table for this process to determine whether the reference was a valid or invalid memory access.
2. If the reference was invalid, we terminate the process. If .it was valid, but we have not yet brought in that page, we now page in the latter.
3. We find a free frame.
4. We schedule a disk operation to read the desired page into the newly allocated frame.
5. When the disk read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory.
6. We restart the instruction that was interrupted by the illegal address trap. The process can now access the page as though it had always been memory.

Therefore, the operating system reads the desired page into memory and restarts the process as though thepage had always been in memory.

The page replacement is used to make the frame free if they are not in used. If no frame is free then otherprocess is called in.

Page Replacement Algorithm

There are many different page replacement algorithms. We evaluate an algorithm by running it on a particular string of memory reference and computing the number of page faults. The string of memory references is called reference string. Reference strings are generated artificially or by tracing a given system and recording the address of each memory reference. The latter choice produces a large number of data.

1. For a given page size we need to consider only the page number, not the entire address.

2. if we have a reference to a page p, then any immediately following references to page p will never causea page fault. Page p will be in memory after the
first reference; the immediately following references will not fault.

Eg:- consider the address sequence

0100, 0432, 0101, 0612, 0102, 0103, 0104, 0101, 0611, 0102, 0103, 0104, 0101, 0610, 0102,

0103, 0104, 0104, 0101, 0609, 0102, 0105
and reduce to 1, 4, 1, 6,1, 6, 1, 6, 1, 6, 1

To determine the number of page faults for a particular reference string and page replacement algorithm,we also need to know the number of page frames available. As the number of frames available increase, the number of page faults will decrease.

FIFO Algorithm

The simplest page-replacement algorithm is a FIFO algorithm. A FIFO replacement algorithm associates with each page the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen. We can create a FIFO queue to hold all pages in memory.

The first three references (7, 0, 1) cause page faults, and are brought into these empty eg. 7, 0, 1, 2, 0, 3,
0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1 and consider 3 frames. This replacement means that the next reference to 0 will fault. Page 1 is then replaced by page 0.

Optimal Algorithm

An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN. It is simply

Replace the page that will not be used
for the longestperiod of time.

Now consider the same string with 3 empty frames.

The reference to page 2 replaces page 7, because 7 will not be used until reference 15, whereas page 0 will be used at 5, and page 1 at 14. The reference to page 3 replaces page 1, as page 1 will be the last of the three pages in memory to be referenced again. Optimal replacement is much better than a FIFO.

The optimal page-replacement algorithm is difficult to implement, because it requires future knowledge of the reference string.

LRU Algorithm

The FIFO algorithm uses the time when a page was brought into memory; the OPT algorithm uses thetime when a page is to be used. In LRU replace the page that has not been used for the longest period of time.

LRU Approximation Algorithms

Some systems provide no hardware support, and other page-replacement algorithm. Many systems provide some help, however, in the form of a reference bit. The reference bit for a page is set, by the hardware, whenever that page is referenced. Reference bits are associated with each entry in the page table Initially, all bits are cleared (to 0) by the operating system. As a user process executes, the bit associated with each page referenced is set (to 1) by the hardware.

Additional-Reference-Bits Algorithm

The operating system shifts the reference bit for each page into the high-order or of its 5-bit byte, shifting the other bits right 1 bit, discarding the low-order bit.

These 5-bit shift registers contain the history of page use for the last eight time periods. If the shift register contains 00000000, then the page has not been

used for eight time periods; a page that is used at least once each period would have a shift register value of11111111.

Second-Chance Algorithm

The basic algorithm of second-chance replacement is a FIFO replacement algorithm. When a page gets a second chance, its reference bit is cleared and its arrival e is reset to the current time.

Enhanced Second-Chance Algorithm

The second-chance algorithm described above can be enhanced by considering troth the reference bitand the modify bit as an ordered pair.

1. (0,0) neither recently used nor modified best page to replace.
2. (0,1) not recently used but modified not quite as good, because the page will need to be written outbefore replacement.
3. (1,0) recently used but clean probably will be used again soon.
4. (1,1) recently used and modified probably will be used again, and write out will be needed beforereplacing it

Counting Algorithms

There are many other algorithms that can be used for page replacement.

- **LFU Algorithm:** The least frequently used (LFU) page-replacement algorithm requires that the page with the smallest count be replaced. This algorithm suffers from the situation in which a page is used heavily during the initial phase of a process, but then is never used again.

- **MFU Algorithm:** The most frequently used (MFU) page-replacement algorithm is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

Page Buffering Algorithm

When a page fault occurs, a victim frame is chosen as before. However, the desired page is read into a free frame from the pool before the victim is written out.

This procedure allows the process to restart as soon as possible, without waiting for the victim page to be written out. When the victim is later written out, its frame is added to the free-frame pool.

When the FIFO replacement algorithm mistakenly replaces a page mistakenly replaces a page that is still in active use, that page is quickly retrieved from the free-frame buffer, and no I/O is necessary. The free-frame buffer provides protection against the relatively poor, but simple, FIFO replacement algorithm.

UNIT VI

Principles of deadlock

To develop a description of deadlocks, which prevent sets of concurrent processes from completing theirtasks.To present a number of different methods for preventing or avoiding deadlocks in a computer system

The Deadlock Problem

A set of blocked processes each holding a resource and waiting to acquire a resource held by another process inthe set

Example

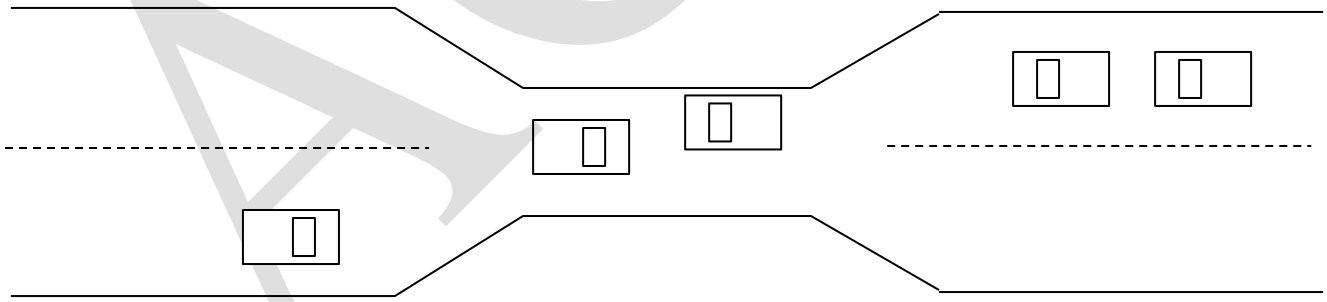
System has 2 disk drives

P_1 and P_2 each hold one disk drive and each needs another oneExample

semaphores *A* and *B*, initialized to 1

P_0	*P1*
wait (A);	wait(B)
wait (B);	wait(A)

Bridge Crossing Example



Traffic only in one direction

Each section of a bridge can be viewed as a resource

If a deadlock occurs, it can be resolved if one car backs up

(preempt resources and rollback)Several cars may have to be backed up if a deadlock occurs
Starvation is possible
Note – Most OSes do not prevent or deal with deadlocks

System Model
Resource types R_1, R_2, . . ., R_m
CPU cycles, memory space, I/O devices Each resource type R_i has W_i instances.
Each process utilizes a resource as follows:
re
qu
est
us
e
rel
eas
e

Deadlock Characterization

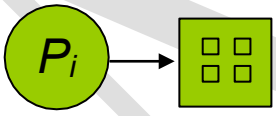
Deadlock can arise if four conditions hold simultaneously
Mutual exclusion: only one process at a time can use a resource
Hold and wait: a process holding at least one resource is waiting to acquire additional resources held by otherprocesses
No preemption: a resource can be released only voluntarily by the process holding it, after that process hascompleted its task
Circular wait: there exists a set $\{P_0, P_1, \ldots, P_0\}$ of waiting processes such that P_0 is waiting for a resource thatis held by P_1, P_1 is waiting for a resource that is held by
$P_2, \ldots, P_{n-1}$ is waiting for a resource that is held by
P_n, and P_0 is waiting for a resource that is held by P_0.
n
Resource-Allocation Graph

A set of vertices V and a set of edges E

V is partitioned into two types:
$P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the processes in the system $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the systemrequest edge – directed edge $P_1 \to R_j$
assignment edge – directed edge $R_j \to P_i$

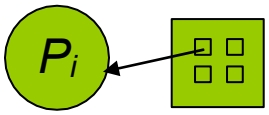
Process 

Resource Type with 4 ins  s

P_i requests instance of R_jn 

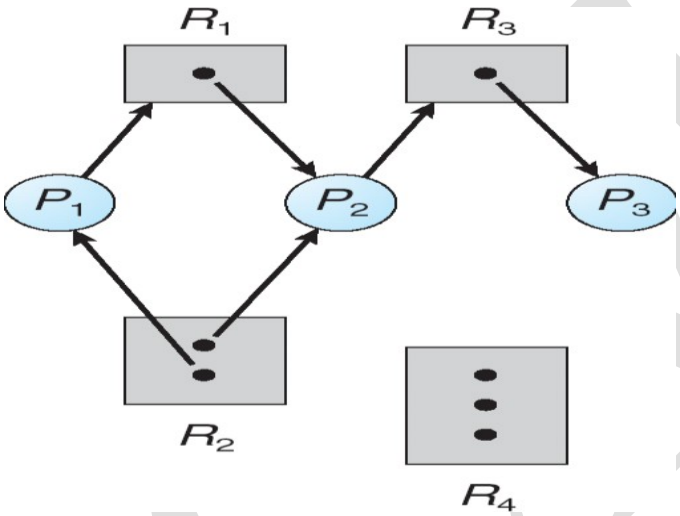
R_j

P_i is holding an instance of R_j

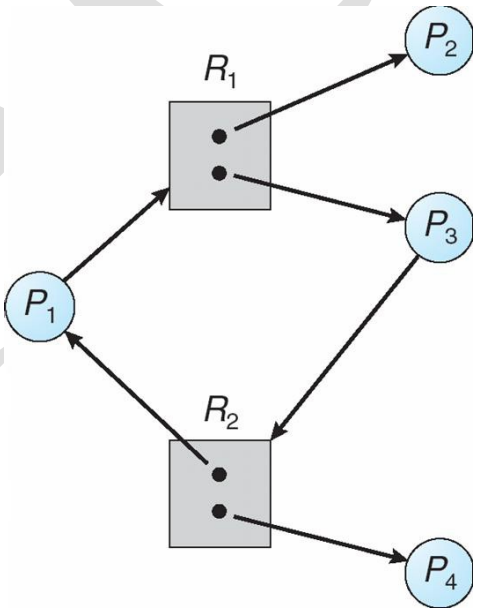


R_j

Example of a Resource Allocation Graph



Graph With A Cycle But No Deadlock



Basic Facts

If graph contains no cycles Þ no deadlocknIf graph contains a cycle Þlif only one instance per resource type,then deadlock

if several instances per resource type, possibility of deadlock

Methods for Handling Deadlocks

Ensure that the system will *never* enter a deadlock statenAllow the system to enter a deadlock state and thenrecovernIgnore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX

Deadlock Prevention

Restrain the ways request can be made

Mutual Exclusion – not required for sharable resources; must hold for nonsharable resources

Hold and Wait – must guarantee that whenever a process requests a resource, it does not hold any otherresources

Require process to request and be allocated all its resources before it begins execution, or allow process torequest resources only when the process has none

Low resource utilization; starvation possible

No Preemption –

If a process that is holding some resources requests another resource that cannot be immediately allocated to it,then all resources currently being held are released

Preempted resources are added to the list of resources for which the process is waiting

Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting **Circular Wait** – impose a total ordering of all resource types, and require that each process requests resourcesin an increasing order of enumeration

Deadlock Avoidance

Requires that the system has some additional *a priori* information available

Simplest and most useful model requires that each process declare the *maximum number* of resources of each type that it may need

The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition

Resource-allocation *state* is defined by the number of available and allocated resources, and the maximum demands of the processes

Safe State

When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state

System is in safe state if there exists a sequence $<P_1, P_2, ..., P_n>$ of ALL the processes is the systems such that for each P_i, the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j, with $j < i$ That is:

If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished

When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate When P_i terminates, P_{i+1} can obtain its needed resources, and so on

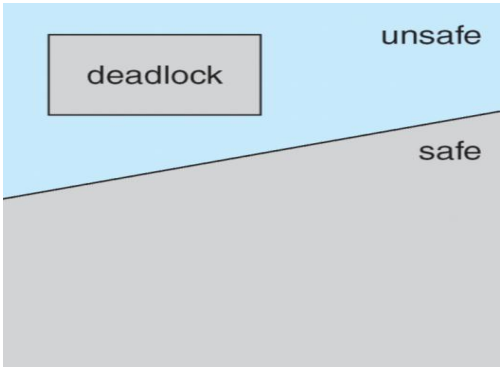
Basic Facts

If a system is in safe state Þ no deadlocks n If a system is in unsafe state Þ possibility of deadlock n Avoidance Þ ensure that a system will never enter an unsafe state.

Safe, Unsafe , Deadlock State

Avoidance algorithms Single instance of a resource type Use a resource-allocation graph

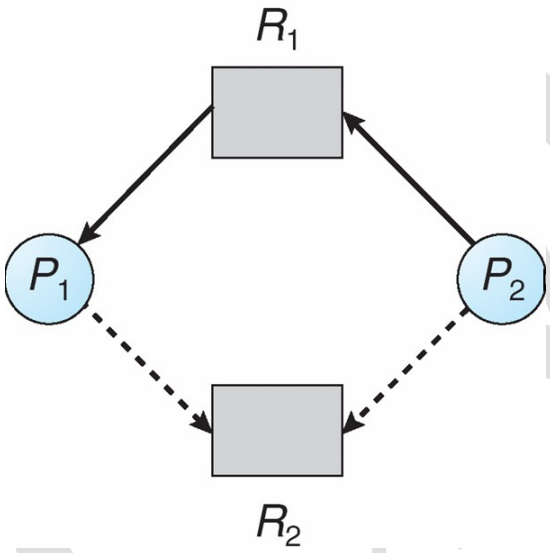
Multiple instances of a resource type Use the banker's algorithm



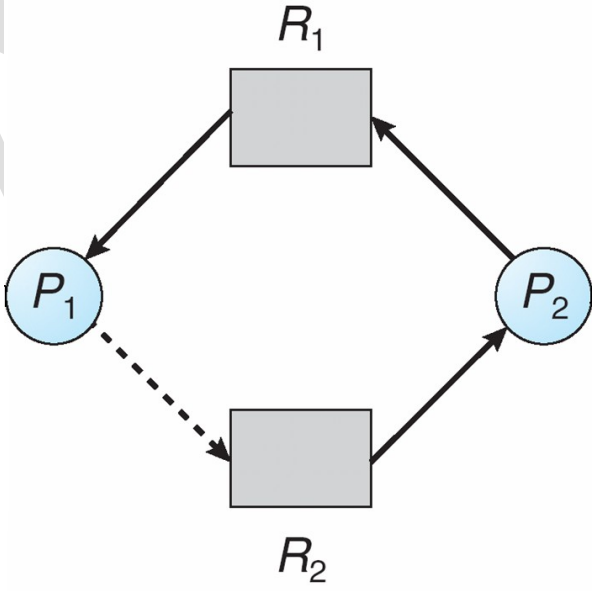
Resource-Allocation Graph Scheme

nClaim edge P_i ® R_j indicated that process P_j may request resource R_j; represented by a dashed linenClaim edgeconverts to request edge when a process requests a resourcenRequest edge converted to an assignment edge when the resource is allocated to the process
nWhen a resource is released by a process, assignment edge reconverts to a claim edgenResources must beclaimed *a priori* in the system

Resource-Allocation Graph



Unsafe State In Resource-Allocation Graph



UNIT VII

FILE SYSTEM INTERFACE

The Concept Of a File

A file is a named collection of related information that is recorded on secondary storage. The information in a file is defined its creator. Many different types of information may be stored in a file.

File attributes:-

A file is named and for the user's convince is referred to by its name. A name is usually a string of characters. One user might create file, where as another user might editthat file by specifying its name. There are different types of attributes.

1) **name:-** the name can be in the human readable form.

2) **type:-** this information is needed for those systems that support different types.

3) **location:-** this information is used to a device and to the location of the file on that device.

4) **size:-** this indicates the size of the file in bytes or words. 5) **protection:-**

6) time,date, and user identifications:-

the information about all files is kept in the directory structure, that also resides on secondary storage.

File operations:-Creating a file:-

Two steps are necessary to create a file first, space in the file system must be foundfor the file. Second , an entry for the new file must be made in the directory. The directory entry records the name of the file and the location in the system.

Writing a file:-

To write a file give the name of the file, the system search the directory to find the location of the file. The system must keep the *writer* pointer to the location in the file where the next write is to take place. The write pointer must be updated whenever a writeoccurs.

Reading a file:- to read from a file, specifies the name of the file and directory is searchfor the associated directory entry, and the system needs to keep *read* pointer to the location in the file where the next read is to take place. Once the read has taken place, read pointer is updated.

Repositioning with in a file:-

The directory is searched for the appropriate entry and the current file position is set togiven value. this is also known as a file seek.

Deleting a file:- to delete a file , we search the directory for the name file. Found that filein the directory entry, we release all file space and erase the directory entry.

Truncate a file:- this function allows all attributes to remain unchanged(except for filelength) but for the file to be reset to length zero.

Appending:- add new information to the end of an existing file .
Renaming:- give new name to an existing file.
Open a file:-if file need to be used, the first step is to open the file, using the *open* system call.
Close:- close is a system call used to terminate the use of an already used file.

File Types:-

A common technique for implementing file type is to include the type as part of the filename. The name is split in to two parts
1) the name 2) and an extension .
the system uses the extension to indicate the type of the file and the type of operations that can be done on that file.

: ACCESSMETHODS:-

There are several ways that the information in the file can be accessed. **1)sequential method 2) direct access method 3) other access methods.** 1)sequential access method:-
the simplest access method is S.A. information in the file is processed in order, one after the other. the bulk of the operations on a file are reads & writes. It is based on a tape model of a file. Fig 10.3
2)Direct access:- or relative access:-
a file is made up of fixed length records, that allow programs to read and write record rapidly in no particular order. For direct access, file is viewed as a numberedsequence of blocks or records. A direct access file allows, blocks to be read & write.So we may read block15, block 54 or write block10. there is no restrictions on the order of reading or writing for a direct access file. It is great useful for immediate access to large amount of information.
The file operations must be modified to include the block number as a parameter.We have read n, where n is the block number.
3)other access methods:-
the other access methods are based on the index for the file. The indexed contain pointers to the various blocks. To find an entry in the file , we first search the index and then use the pointer to access the file directly and to find the desired entry. Withlarge files. The index file itself, may become too large to be kept in memory. One solution is to create an index for the index file. The primary index file would contain pointers to secondary index files which would point to the actual data iteams

Directory Structures:-

operations that are be on a directory (read in text book)
single level directory:-

the simple directory structure is the single level directory. All files are contained in the same directory. Which is easy to understand. Since all files are in same directory,they must have unique names.

In a single level directory there is some limitations. When the no.of files increases or when there is more than one user some problems can occurs. If the no.offiles increases, it becomes difficult to remember the names of all the files.

FIG 10.7 **Two-level directory:-**

The major disadvantages to a single level directory is the confusion of file names between different users. The standard solution is to create separate directory for eachuser.

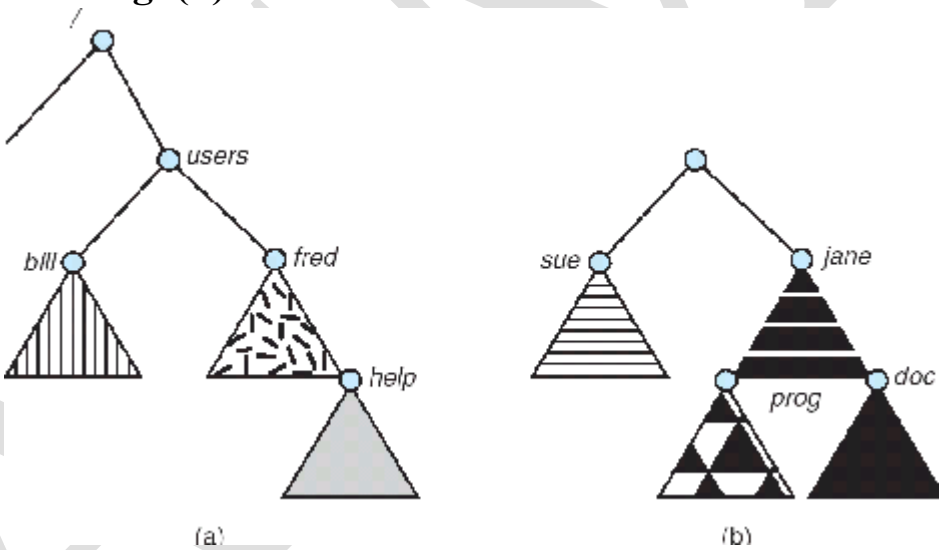
In 2-level directory structure, each user has her own user file directory(ufd). Each ufdhas a similar structure, the user first search the master file directory . the mfd is indexed by user name and each entry point to the ufd for that user.fig 10.8

:File System Mounting

A file system must be **mounted** before it can be accessed

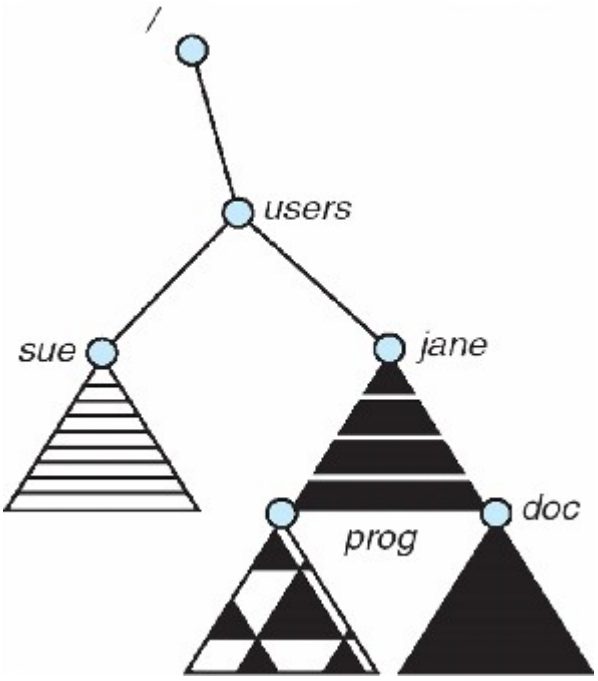
A unmounted file system (i.e. Fig. 11-11(b)) is mounted at a **mount point**

(a) Existing. (b) Unmounted Partition



(b)

Mount Point



FILE SYSTEM IMPLEMENTATION

:File allocation methods:-

There are 3 major methods of allocating disk space.

1) **Contiguous allocation:-**

1)The contiguous allocation method requires each file to occupy a set ofcontiguous block on the disk.

2)Contiguous allocation of a file is defined by the disk address and length ofthe first block. If the file is 'n' block long and starts at location 'b' , then it occupies blocks b,b+1,b+2,…..,b+n-1;

3)The directory entry for each file indicates the address of the starting blockand length of the area allocated for this file. Fig **11.3**

4)Contiguous allocation of file is very easy to access. For the *sequentialaccess* , the file system remembers the disk address of the last block referenced and, when necessary read next block. For *direct access* to block 'i' of a file that starts at block 'b' , we can immediately access blockb+i. Thus both sequential and direct access can be supported by contagious allocation.

5)One difficulty with this method is finding space for a new file.

6)Also there are many problems with this method

a) **external fragmentation:-** files are allocated and deleted , thefree disk space is broken in to little pieces. The E.F exists when free space is broken in to chunks(large piece) and these chunksare not sufficient for a request of new file. There is a solution for E.F i.e compaction. All free space compact into one contiguous space. But the cost of compaction is time.

b) Another problem is determining how much space is needed for afile. When file is created the creator must specifies the size of
that file. This becomes to big problem. Suppose if we allocatetoo little space to a file , some times it may not sufficient.
Suppose if we allocate large space some times space is wasted.
c) Another problem is if one large file is deleted, that large space isbecomes to empty. Another file is loaded in to that space whose size is very small then some space is wasted . that wastage of space is called internal fragmentation.

2) **Linked allocation:-**

1) Linked allocation solves all the problems of contagious allocation. With linked allocation , each file is a linked list of disk blocks, the disk block may be scattered any where on the disk.
2) The directory contains a pointer to the first and last blocks of the file. **Fig11.4**
Ex:- a file have five blocks start at block 9, continue at block 16,then block 1, block 10 and finally block 25. each block contains a ponter to the next block. These pointers are not available to the user.
3) To create a new file we simply creates a new entry in directory. With linked allocation, each directory entry has a pointer to the first disk block of the file.
3) There is no external fragmentation with linked allocation. Also there is no need to declare the size of a file when that file is created. A file can continue togrows as long as there are free blocks.

4) But it have disadvantage. The major problem is that it can be used only for sequential access-files.
5) To find the I th block of a file , we must start at the beginning of that file, and follow the pointers until we get to the I th block. It can not support the direct access.
6) Another disadvantage is it requires space for the pointers. If a pointer requires4 bytes out of 512 byte block, then 0.78% of disk is being used for pointers, rather than for information.
7) The solution to this problem is to allocate blocks in to multiples, called clusters and to allocate the clusters rather than blocks.
8) Another problem is reliability. The files are linked together by pointers scattered all over the disk what happen if a pointer were lost or damaged.**FAT(file allocation table):-**

An important variation on the linked allocation method is the use of a file allocation table.
The table has one entry for each disk block, and is indexed by block number. The

FAT is used much as is a linked list.
The directory entry contains the block number of the first block of the file. The table entry contains the block number then contains the block number of the next block in the file. This chain continuous until the last block, which has a special endof file values as the table entry. Unused blocks are indicated by a '0' table value. Allocation a new block to a file is a simple. First finding the first 0-value table entry, and replacing the previously end of file value with the address of the new block. The 0 is then replaced with end of file value.

Fig 11.5

3)Indexed allocation:-

1) linked allocation solves the external fragmentation and size declaration problems of contagious allocation. How ever in the absence of a FAT , linked allocation can not support efficient direct access.

2) The pointers to the blocks are scattered with the blocks themselves all over the disk and need to be retrieved in order.

3) Indexed allocation solves this problem by bringing all the pointers together into one location i.e *the index block.*

4) Each file has its own index block ,which is an array of disk block addresses. The I th entry in the index block points to the ith block of the file.

5) The directory contains the address of the index block. **Fig 11.6**

To read the ith block we use the pointer in the ith index block entry to findand read the desired block.

6) When the file is created, all pointers in the index block are set to nil. When the ith block is first written, a block is obtained from the free space manager, and its address is put in the ith index block entry.

7) It supports the direct access with out suffering from external fragmentation,but it suffer from the wasted space. The pointer overhead of the index block is generally greater than the pointer over head of linked allocation.

:Free space management:-

1) to keep track of free disk space, the system maintains a free space list. The free space list records all disk blocks that are free.

2) To create a file we search the free space list for the required amount of space, andallocate that space to the new file. This space is then removed from the free space list.

3) When the file is deleted , its disk space is added to the free space list.There are many methods to find the free space.

1) **bit vector:-**

The free space list is implemented as a bit map or bit vector. Each block is represented by 1 bit. If the block is free the bit is 1 if the block is allocated the bit

is 0.

Ex:- consider a disk where blocks 2,3,4,5,8,9,10,11,12,13,17,18,25, are free and rest of blocks are allocated the free space bit map would be

00111100111111000110000001000……..

the main advantage of this approach is that it is relatively simple and efficient to find the first free block or 'n' consecutive free blocks on the disk

2) **Linked list:-**

Another approach is to link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory. This first block contain a pointer to the next free disk block, and so on.

How ever this scheme is not efficient to traverse the list, we must read each block, which requires I/O time.

Disk space is also wasted to maintain the pointer to next free space.

3) **Grouping:-**

Another method is store the addresses of 'n' free blocks in the first free block. The first (n-1) of these blocks are actually free. The last block contains the addresses of another 'n' free blocks and so on. **Fig 11.8**

Advantages:- the main advantage of this approach is that the addresses of a large no.of blocks can be found quickly.

4) **Counting:-**

Another approach is counting. Generally several contiguous blocks may be allocated or freed simultaneously. Particularly when space is allocated with the contiguous allocation algorithm rather than keeping a list of 'n' free disk address. We can keep the address of first free block and the number 'n' of free contiguous blocks that follow the first block. Each entry in the free space list then consists of a disk address and a count.

:Directory Implementation:-

1) Linear list:-

1) the simple method of implement ting a directory is to use a linear list of file names with pointers to the data blocks.

2) A linear list of directory entries requires a linear search to find a particular entry.

3) This method is simple to program but is time consuming to execute.

4) To create a new file, we must first search the directory to be sure that no existing file has the same name. Then, we add a new entry at the end of the directory.

5) To delete a file we search the directory for the named file, then release the space allocated to it.

6) To reuse directory entry, we can do one of several things.

7) We can mark the entry as unused or we can attach it to a list of freedirectory entries.

Disadvantage:- the disadvantage of a linear list of directory entries is thelinear search to find a file.

UNIT VIII

MASS-STORAGE STRUCTURE

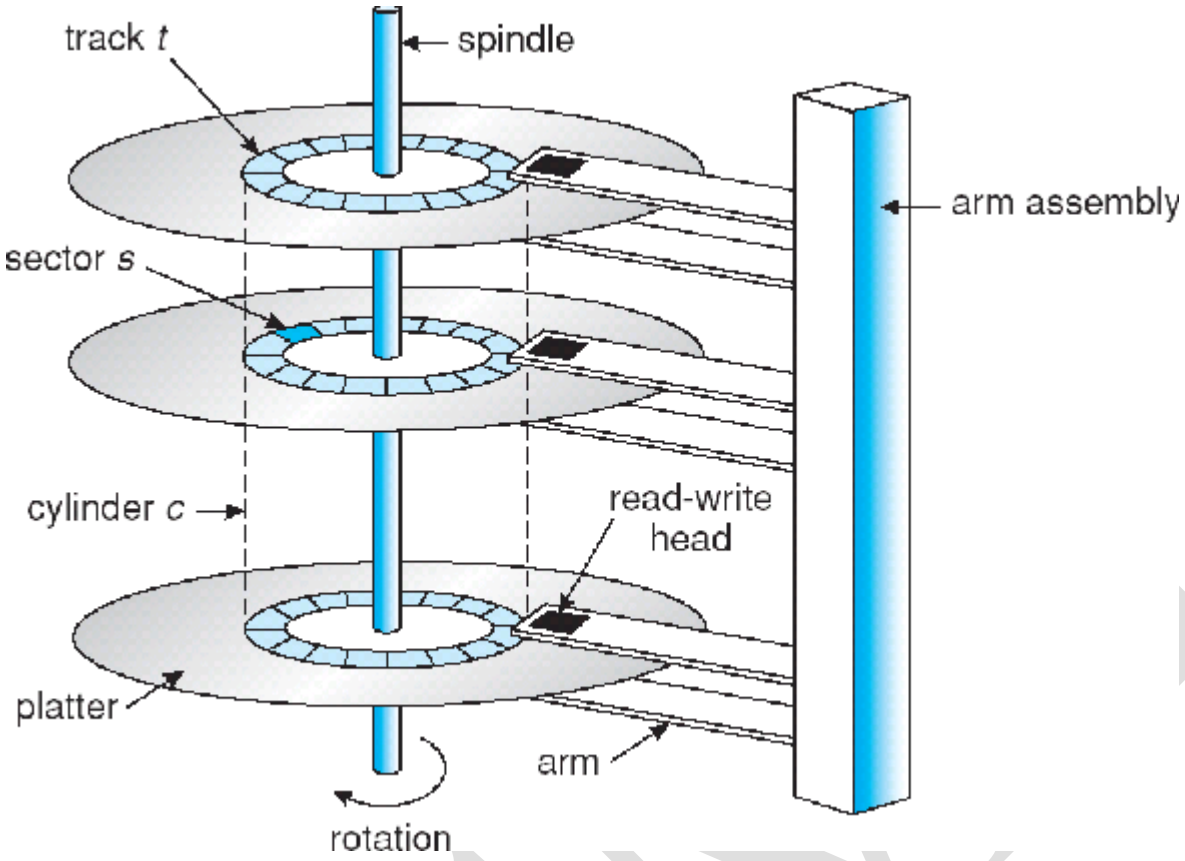
Mass-Storage Systems

nDescribe the physical structure of secondary and tertiary storage devices andthe resulting effects on the uses of the devicesnExplain the performance characteristics of mass-storage devicesnDiscuss operating-system services provided for mass storage, including RAID and HSM

:Overview of Mass Storage Structure

Magnetic disks provide bulk of secondary storage of modern computersDrives rotate at 60 to 200 times per second
Transfer rate is rate at which data flow 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 results from disk head making contact with the disk surface
That's bad
Disks can be removable
Drive attached to computer via I/O bus
Busses vary, including EIDE, ATA, SATA, USB, Fibre Channel, SCSI
Host controller in computer uses bus to talk to disk controller built into drive or storagearray

Moving-head Disk Mechanism



Magnetic tape
Was early secondary-storage medium
Relatively permanent and holds large quantities of dataAccess time slow
Random access ~1000 times slower than disk
Mainly used for backup, storage of infrequently-used data, transfer medium between systems
Kept in spool and wound or rewound past read-write headOnce data under head, transfer rates comparable to disk 20-200GB typical storage
Common technologies are 4mm, 8mm, 19mm, LTO-2 and SDLT

:Disk Structure

Disk drives are addressed as large 1-dimensional arrays of logical blocks, where the logical block is the smallest unit of transfernThe 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially
Sector 0 is the first sector of the first track on the outermost cylinder
Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost

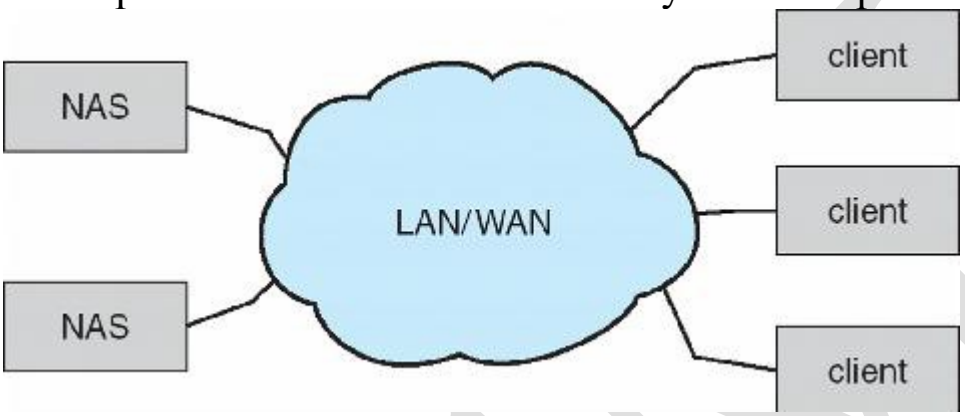
8.3:Disk Attachment

Host-attached storage accessed through I/O ports talking to I/O busses SCSI itself is a bus, up to 16 devices on one cable, SCSI initiator requestsoperation and SCSI targets perform tasks
Each target can have up to 8 logical units (disks attached to device controllerFC

is high-speed serial architecture
Can be switched fabric with 24-bit address space – the basis of storage area networks (SANs) in which many hosts attach to many storage units
Can be arbitrated loop (FC-AL) of 126 devices

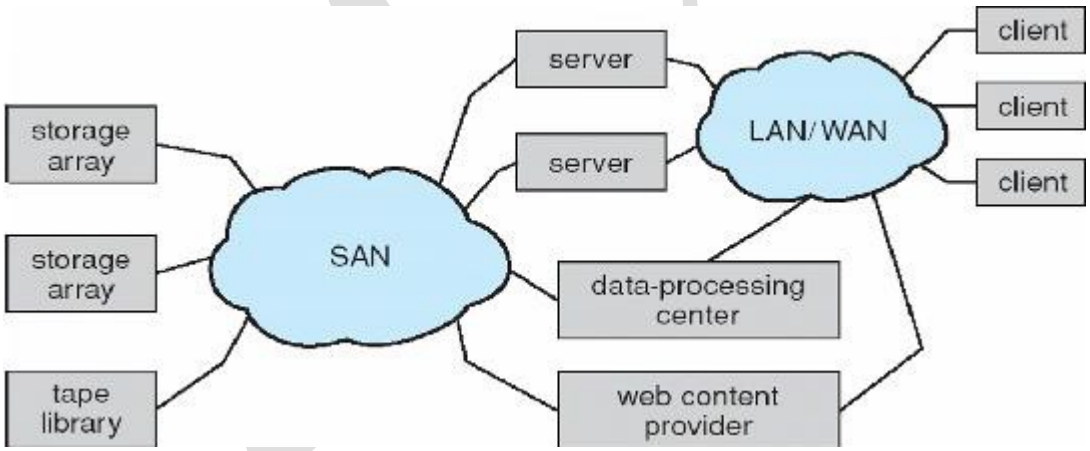
Network-Attached Storage

Network-attached storage (NAS) is storage made available over a networkrather than over a local connection (such as a bus)
NFS and CIFS are common protocols
Implemented via remote procedure calls (RPCs) between host and storageNew iSCSI protocol uses IP network to carry the SCSI protocol



Storage Area Network

Common in large storage environments (and becoming more common)Multiple hosts attached to multiple storage arrays – flexible



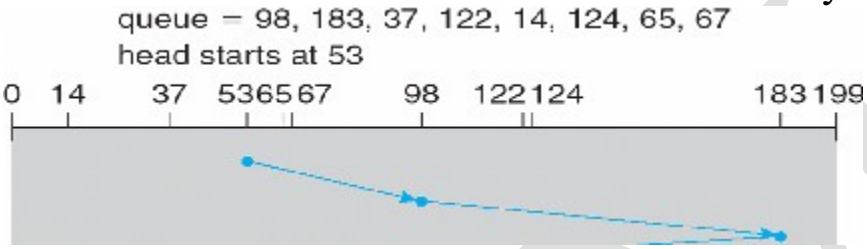
:Disk Scheduling

The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth
Access time has two major components
Seek time is the time for the disk are to move the heads to the cylindercontaining the desired sector

Rotational latency is the additional time waiting for the disk to rotate the desired sector to the disk head
Minimize seek time
Seek time » seek distance
Disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer
Several algorithms exist to schedule the servicing of disk I/O requests nWe illustrate them with a request queue (0-199)
98, 183, 37, 122, 14, 124, 65, 67
Head pointer 53

FCFS

Illustration shows total head movement of 640 cylinders



SSTF

Selects the request with the minimum seek time from the current head position
SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests
nIllustration shows total head movement of 236 cylinders

SCAN

The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.nSCAN algorithm Sometimes called the elevator algorithm

Illustration shows total head movement of 208 cylinders

C-SCAN

Provides a more uniform wait time than SCAN
The head moves from one end of the disk to the other, servicing requests as it goes
When it reaches the other end, however, it immediately returns to the beginningof the disk, without servicing any requests on the return trip
Treats the cylinders as a circular list that wraps around from the last cylinder to the first one

C-LOOK

Version of C-SCAN
Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk**Selecting a**

Disk-Scheduling Algorithm

SSTF is common and has a natural appeal

SCAN and C-SCAN perform better for systems that place a heavy load on the disk

Performance depends on the number and types of requests

Requests for disk service can be influenced by the file-allocation method The disk-scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm if necessary

Either SSTF or LOOK is a reasonable choice for the default algorithm

Disk Management

Low-level formatting, or physical formatting — Dividing a disk into sectors that the disk controller can read and write

To use a disk to hold files, the operating system still needs to record its own data structures on the disk

Partition the disk into one or more groups of cylinders Logical formatting or "making a file system"

To increase efficiency most file systems group blocks into clusters

- Disk I/O done in blocks
- File I/O done in clusters Boot block initializes system

The bootstrap is stored in ROM Bootstrap loader program

Methods such as sector sparing used to handle bad blocks

Tertiary Storage Devices

Low cost is the defining characteristic of tertiary storage Generally, tertiary storage is built using removable media Common examples of removable media are floppy disks and CD-ROMs; other types are available

Removable Disks

Floppy disk — thin flexible disk coated with magnetic material, enclosed in a protective plastic case Most floppies hold about 1 MB; similar technology is used for removable disks that hold more than 1 GB

Removable magnetic disks can be nearly as fast as hard disks, but they are at a greater risk of damage from exposure