ERODE ARTS AND SCIENCE COLLEGE (AUTONOMOUS)

ODD SEMESTER

OPERATING SYSTEM

SUB STAFF: M .Divya, M.Sc., M.Phil., CLASS: II B.Sc CS

HOD SIGN:

UNIT I:

Introduction: What is an Operating System? – Batch Systems – Time Sharing Systems – Personal Computer Systems – Parallel Systems – Real Time Systems – Distributed Systems – Computer System Operation – I/O interrupts. Operating System Structures: System Components – System Services – System Structure.

UNIT II:

Process Management: Process Concept – Process Scheduling- Operations on Process. CPU Scheduling: Basic Concepts – Scheduling Criteria – Scheduling Algorithms – Multiprocessor Scheduling – Real Time Scheduling – Threads.

UNIT III:

Process Synchronization: Background- Critical – Section Problem – Two task solutions- Synchronization Hardware – Semaphores.Deadlocks: System Model – Deadlock Characterization – Methods for Handling Deadlocks –– Deadlock Prevention - Deadlock Avoidance – Deadlock Detection.

UNIT IV:

Memory Management: Background – Swapping – Continuous Memory Allocation- Paging- Segmentation. Virtual Memory: Background – Demand Paging – Page Replacement.

UNIT V:

File Systems: File Concept – File Attributes – File Operations – file Types – Access methods – Directory Structure – Single Level Directory – Two Level Directory – Tree Structure Directories – Protection – Types of Access – File System Structure – Allocation methods. Mass Storage Structure: Disk Structure – Disk Scheduling – Disk Management.

TEXT BOOK:

1. Silberschatz, Galvin, Gagne, “Operating System Concepts”, Replika Press Pvt. Ltd., Sixth
Edition, 2002.
Unit I : Chapter 1,2,3 Unit II : Chapter 4,5 Unit III : Chapter 7,8 Unit IV : Chapter 9,10
Unit V : Chapter 11,13

REFERENCE BOOKS:

1. H.M.Deitel, " Operating System", Addison Wesley Publication, Second Edition.

UNIT-I

INTRODUCTION TO OPERATING SYSTEM:

What is an Operating System?

A program that acts as an intermediary between a user of a computer and the computer hardware

Operating system goals:

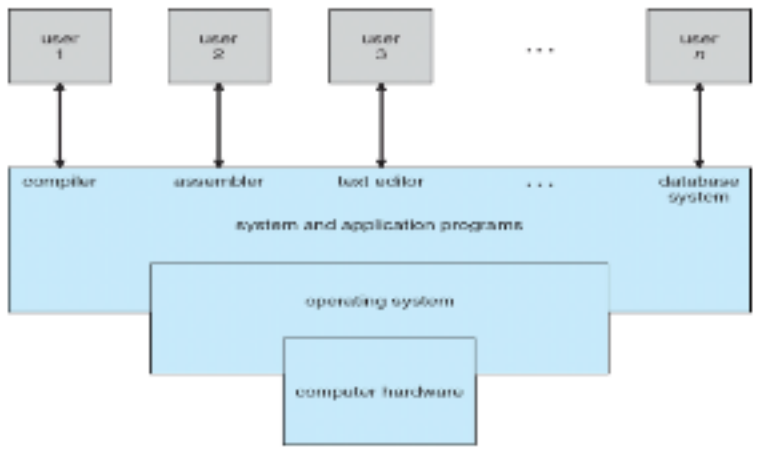
a. Execute user programs and make solving user problems easier
b. Make the computer system convenient to use
c. Use the computer hardware in an efficient manner

Computer system components:

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 computing problems of the users
  - Word processors, compilers, web browsers, database systems, video games
- Users
  - People, machines, other computers

Four Components of a Computer System



Functions of Operating System

1. It boots the computer
2. It performs basic computer tasks e.g. managing the various peripheral devices e.g. mouse, keyboard
3. It provides a user interface, e.g. command line, graphical user interface (GUI)
4. It handles system resources such as computer's memory and sharing of the central processing unit(CPU) time by various applications or peripheral devices.
5. It provides file management which refers to the way that the operating system manipulates, stores, retrieves and saves data.
6. Error Handling is done by the operating system. It takes preventive measures whenever required to avoid errors.

Operating System Management Tasks

1. **Processor management** which involves putting the tasks into order and pairing them into manageable size before they go to the CPU.
2. **Memory management** which coordinates data to and from RAM (random-access memory) and determines the necessity for virtual memory.
3. **Device management** which provides interface between connected devices.
4. **Storage management** which directs permanent data storage.
5. **Application** which allows standard communication between software and your computer.
6. **User interface** which allows you to communicate with your computer.

TYPES OF OPERATING SYSTEM:

1. **Batch Operating Systems**

2. Time Sharing Systems
3. **Personal Computer Systems**
4. **Parallel Systems**
5. **Distributed systems**
6. **Real time Systems**

1.1 BATCH OPERATING SYSTEM

Batch processing:

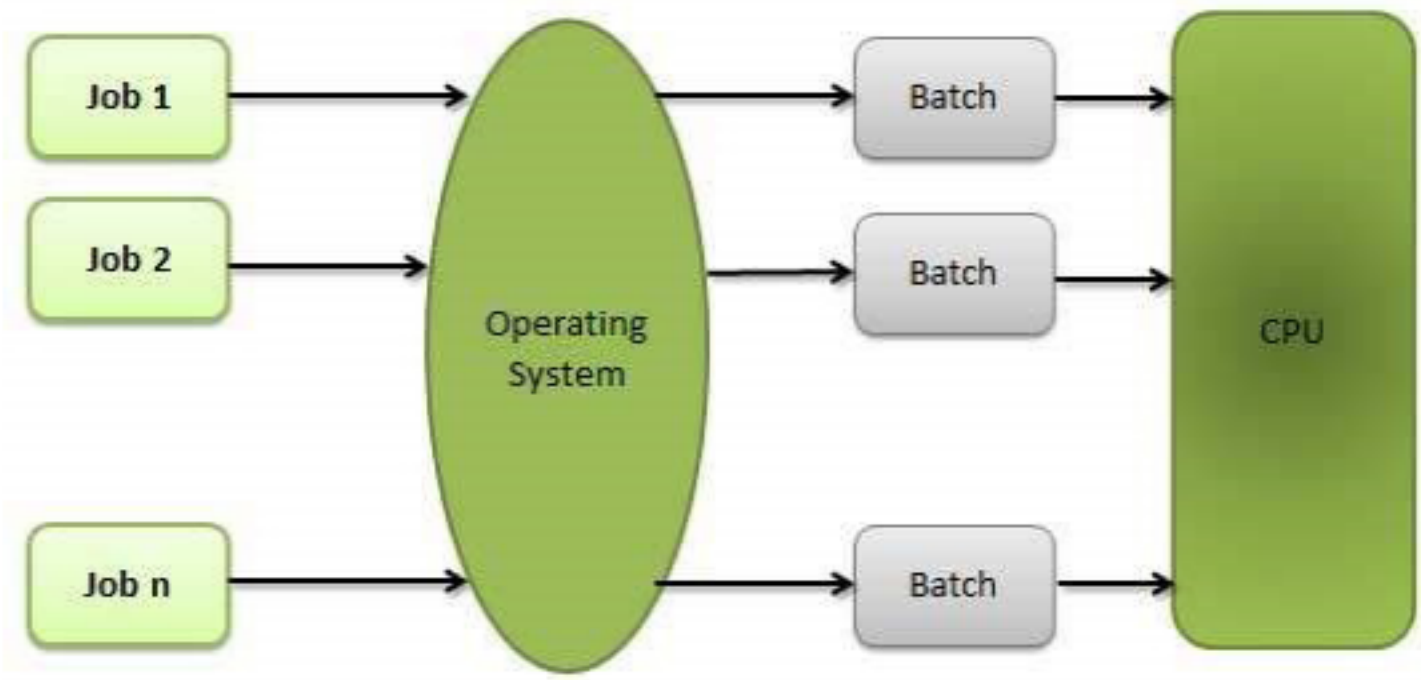
★ Batch means set of jobs with similar needs

Example:

Batch processing is a technique in which an Operating System collects the programs and data together in a batch before processing starts.

An operating system does the following activities related to batch processing:

- The OS defines a job which has predefined sequence of commands, programs and data as a single unit.
- The OS keeps a number a jobs in memory and executes them without any manual information.
- Jobs are processed in the order of submission, i.e., first come first served fashion.
- When a job completes its execution, its memory is released and the output for the job gets copied into an output spool for later printing or processing.



Advantages

- Batch processing takes much of the work of the operator to the computer.
- Payroll run of company
- Gas a electricity bill produced by batch system
- Increased performance as a new job gets started as soon as the previous job is finished, without any manual intervention.

Disadvantages

- Difficult to debug program.
- No interaction between user and computer
- No priority
- Cpu is often idle
- Speed is slow
- A job could enter an infinite loop.
- Due to lack of protection scheme, one batch job can affect pending jobs.

Example

- **CARD READER**
- It is a machine.
- Card reader is also called credit card reader
- It is a electronic device that allows you to process debit and credit payments.
- Safer for processing cash as well as managing cash.

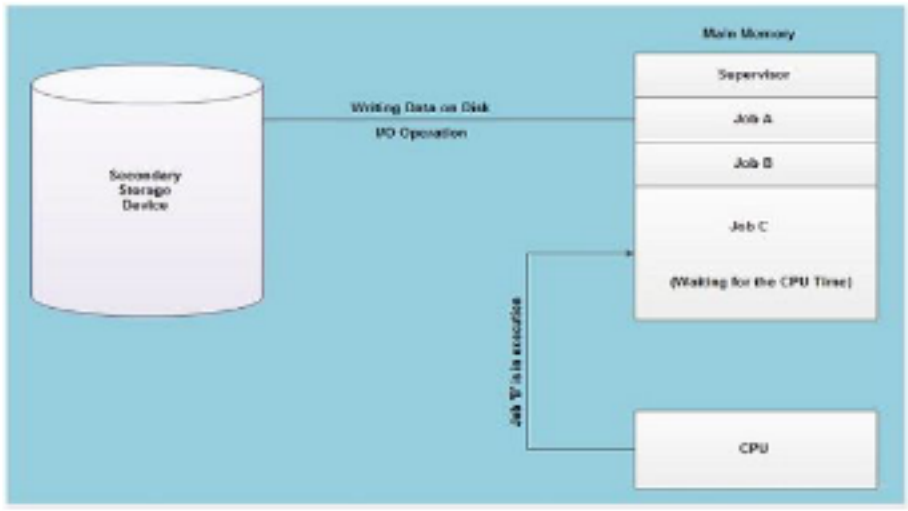
What is **Spooling?**

- ★ Spooling is a process in which data is temporarily held to be used and executed by a device, program or the system.
- ★ Card reader request send to the disk and disk will be processed.
- ★ Card will be scanned for images, signature and maintaining table for customer details in Operating system.
- ★ Job will be executed after processing sent the satisfaction to the disk finally job is completed then output printed.
- ★ This form of processing is called spooling.

Multiprogramming batch operating system:

- Sharing the processor, when two or more programs reside in memory at the same time, is referred as **multiprogramming**.
- Multiprogramming assumes a single shared processor.
- Multiprogramming increases CPU utilization by organizing jobs so that the CPU always has one to execute.

The following figure shows the memory layout for a multiprogramming system.



In OS do the following activities related to multiprogramming.

- The operating system keeps several jobs in memory at a time.
- This set of jobs is a subset of the jobs kept in the job pool.
- The operating system picks and begins to execute one of the jobs in the memory.
- Multiprogramming operating systems monitor the state of all active programs and system resources using memory management programs to ensure that the CPU is never idle, unless there are no jobs to process.

Advantages

- High and efficient CPU utilization.
- User feels that many programs are allotted CPU almost simultaneously.

Disadvantages

- CPU scheduling is required.
- To accommodate many jobs in memory, memory management is requirement.

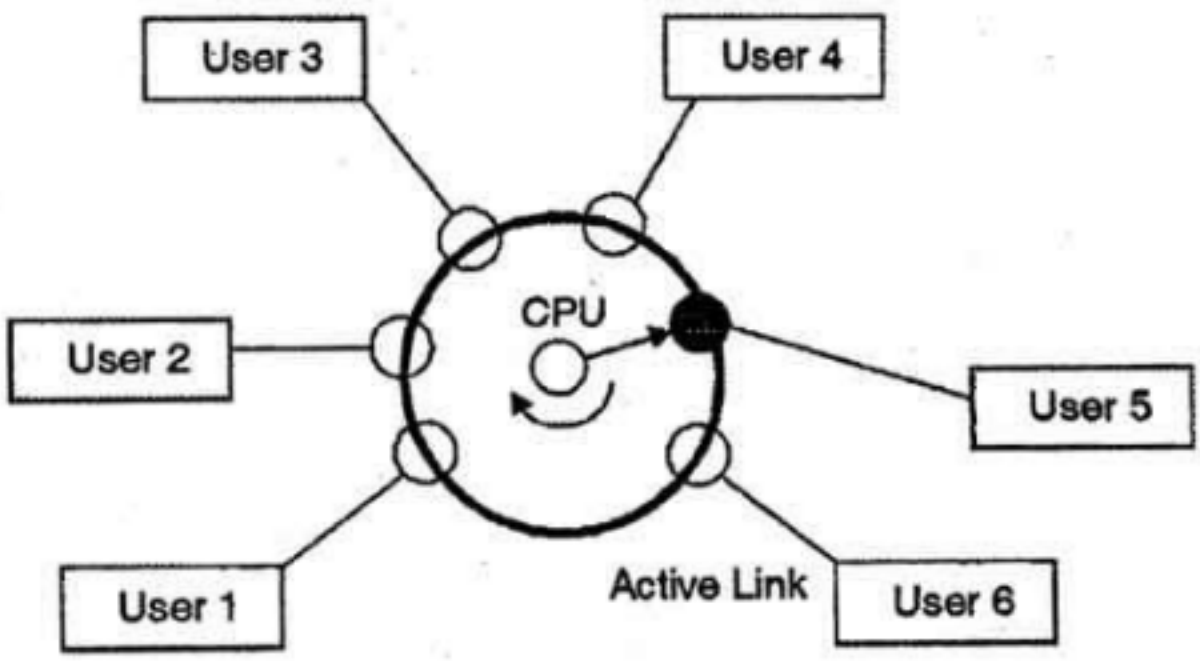
1.2 TIME SHARING SYSTEM

a. Time-sharing is a technique which enables many people, located at various terminals, to use a particular computer system at the same time.

b. Time-sharing or multitasking is a logical extension of multiprogramming.

c. Processor's time which is shared among multiple users simultaneously is termed as time-sharing.

d. The main difference between Multiprogrammed Batch Systems and Time-Sharing Systems is that in case of Multiprogrammed batch systems, the objective is to maximize processor use, whereas in Time-Sharing Systems, the objective is to minimize response time.



- Multiple jobs are executed by the CPU by switching between them, but the switches occur so frequently.
- Thus, the user can receive an immediate response. For example, in a transaction processing, the processor executes each user program in a short burst or quantum of computation.

- That is, if **n** users are present, then each user can get a time quantum. When the user submits the command, the response time is in few seconds at most.
- The operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time.
- Computer systems that were designed primarily as batch systems have been modified to time-sharing systems.

Advantages of Timesharing operating systems are as follows –

- Provides the advantage of quick response.
- Avoids duplication of software.
- Reduces CPU idle time.

Disadvantages of Time-sharing operating systems are as follows –

- Problem of reliability.
- Question of security and integrity of user programs and data.
- Problem of data communication.

1.3 PERSONAL COMPUTER SYSTEMS:

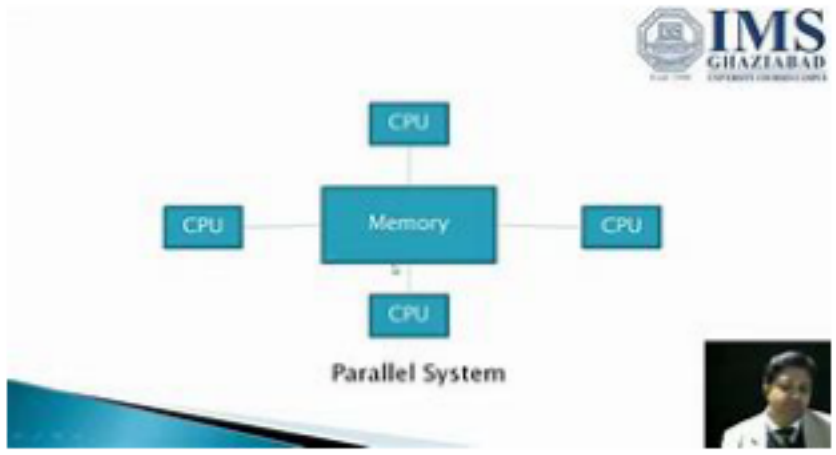
Personal computer appeared in 1970's. They are microcomputers that are smaller & less expensive than mainframe systems.

- Instead of maximizing CPU & peripheral utilization, the systems opt for maximizing user convenience & responsiveness.
- Single user systems, portable.
- I/O devices - keyboards, mice, display screens,small printers.
- Laptops and palmtops, Smart cards, Wireless devices.
- Single user systems may not need advanced CPU utilization or protection features

E.g. : windows 98, windows 2000,

1.5 Parallel Operating Systems:

- Parallel operating systems are a type of computer processing platform that breaks large tasks into smaller pieces that are done at the same time in different places and by different mechanisms
- Parallel os are interface between parallel computers and applications that are executed on them
- They transfer hardware capabilities into concepts usuable by programming language.
- Parallel os able to use software to manage all different resources of computer running is parallel.
- It is also allows user to directly interface with all the computers in network.
- System said to be parallel system which multiple processor have dirct access to shared memory which from a common address space.



Advantages

a. Do multiple things
b. Cost saving

Disadvantages

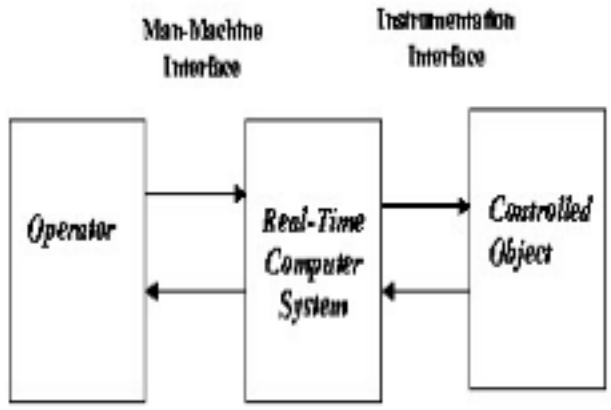
- Programmer responsibility for correct access of global.

1.6 REAL TIME OPERATING SYSTEM

A real-time system is defined as a data processing system in which the time interval required to process and respond to inputs is so small that it controls the environment.

The time taken by the system to respond to an input and display of required updated information is termed as the **response time**.

So in this method, the response time is very less as compared to online processing.



Real-time systems are used when there are rigid time requirements on the operation of a processor or the flow of data and real-time systems can be used as a control device in a dedicated application.

A real-time operating system must have well-defined, fixed time constraints, otherwise the system will fail. For example, scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, air traffic control systems, etc.

There are two types of real-time operating systems:

1. Hard real-time systems
2. Soft real-time systems

Hard real-time systems

Hard real-time systems guarantee that critical tasks complete on time. In hard real-time systems, secondary storage is limited or missing and the data is

stored in ROM. In these systems, virtual memory is almost never found.

Soft real-time systems

Soft real-time systems are less restrictive. A critical real-time task gets priority over other tasks and retains the priority until it completes.

Soft real-time systems have limited utility than hard real-time systems. For example, multimedia, virtual reality, Advanced Scientific Projects like undersea exploration and planetary rovers, etc.

1.7 DISTRIBUTED OPERATING SYSTEM

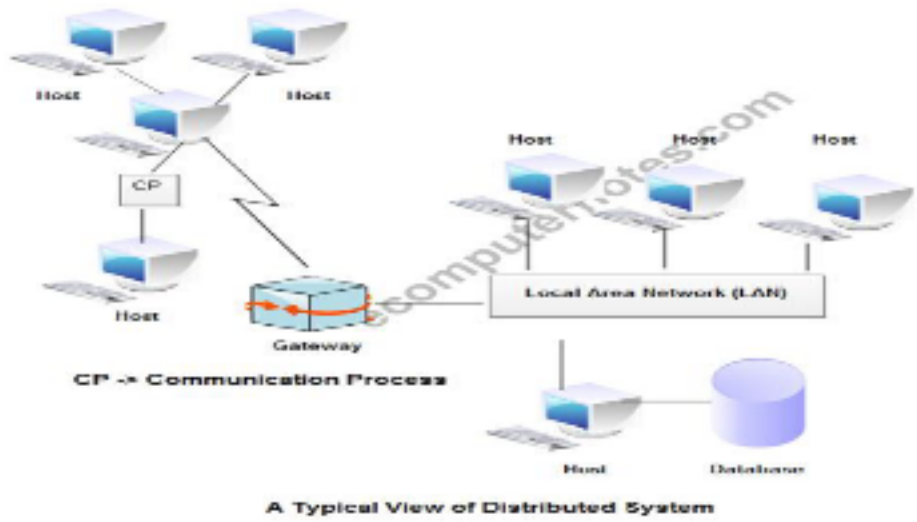
Distributed systems use multiple central processors to serve multiple real-time applications and multiple users.

Data processing jobs are distributed among the processors accordingly.

The processors communicate with one another through various communication lines (such as high-speed buses or telephone lines).

These are referred as **loosely coupled systems** or distributed systems.

Processors in a distributed system may vary in size and function. These processors are referred as sites, nodes, computers, and so on.



A Typical View of Distributed System

Advantages :

The **advantages** of distributed systems are as follows –

- With resource sharing facility, a user at one site may be able to use the resources available at another.
- Speedup the exchange of data with one another via electronic mail.
- If one site fails in a distributed system, the remaining sites can potentially

continue operating.

- Better service to the customers.
- Reduction of the load on the host computer.
- Reduction of delays in data processing.

Disadvantages:

- Security problem due to sharing
- Some messages can be lost in the network system
- expensive
- Overloading is another problem in distributed operating systems
- performance become slow
- The databases in network operating is difficult to administrate then single user system.

Computer-System Operation

★ I/O devices and the CPU can execute concurrently.

★ Each device controller is in charge of a particular device type.

★ Each device controller has a local buffer.

★ CPU moves data from/to main memory to/from local buffers

★ I/O is from the device to local buffer of controller.

★ Device controller informs CPU that it has finished its

★ Operation by causing an *interrupt*.

Operating-System Structures

Common System Components

- ★ Process Management
- ★ Main Memory Management
- ★ File Management
- ★ I/O System Management
- ★ Secondary Management
- ★ Networking
- ★ Protection System
- ★ Command-Interpreter System

Process Management

- ★ A *process is a program in execution.*
- ★ *A process needs* certain resources, including CPU time, memory, files, and I/O devices, to accomplish its task.
- ★ The operating system is responsible for the following activities in connection with process management.
- ★ Process creation and deletion.
- ★ process suspension and resumption.
- ★ Provision of mechanisms for:

✔ Process synchronization

✔ Process communication

Main-Memory Management

- ★ Memory is a large array of words or bytes, each with its own address. It is a repository of quickly accessible data shared by the CPU and I/O devices.
- ★ Main memory is a volatile storage device. It loses its contents in the case of system failure.

★ The operating system is responsible for the following activities in connections with memory management:

✦ Keep track of which parts of memory are currently being used and by whom.

✦ Decide which processes to load when memory space becomes available.

✦ Allocate and deallocate memory space as needed.

File Management

★ A file is a collection of related information defined by its creator. Commonly, files represent programs (both source and object forms) and data.

★ The operating system is responsible for the following activities in connections with file management:

✦ File creation and deletion.

✦ Directory creation and deletion.

✦ Support of primitives for manipulating files and directories.

✦ Mapping files onto secondary storage.

✦ File backup on stable (nonvolatile) storage media

I/O System Management

The I/O system consists of:

✦ A buffer-caching system

✦ A general device-driver interface

✦ Drivers for specific hardware devices

Secondary-Storage Management

★ Main memory (*primary storage) is volatile and too* small to accommodate all data and programs permanently, the computer system must provide *secondary storage to back up main memory.*

★ Most modern computer systems use disks as the principle on-line storage medium, for both programs and data.

★ The operating system is responsible for the following activities in connection with disk management:

✦ Free space management

✦ Storage allocation

✦ Disk scheduling

Protection System

- *Protection refers to a mechanism for controlling access* by programs, processes, or users to both system and user resources.
- The protection mechanism must:

✦Distinguish between authorized and unauthorized usage.

✦ specify the controls to be imposed.

✦ provide a means of enforcement.

Command-Interpreter System

★ Many commands are given to the operating system by

★ control statements which deal with:

✦ process creation and management

✦ I/O handling

✦ secondary-storage management

✦ main-memory management

✦ file-system access

✦ protection

✦ networking

★ The program that reads and interprets control statements is called variously:

✦ command-line interpreter

✦ shell (in UNIX)

★ Its function is to get and execute the next command statement

Operating System Services

- **Program execution :**
  System capability to load a program into memory and to run it.
- **I/O operations:**
  Since user programs cannot execute I/O operations directly, the operating system must provide some means to perform I/O.
- **File-system manipulation :**
  Program capability to read, write, create, and delete files.
- **Communications:**
  Exchange of information between processes executing either on the same computer or on different systems tied together by a network. Implemented via *shared memory or message passing.*
- **Error detection:**
  Ensure correct computing by detecting errors in the CPU and memory hardware, in I/O devices, or in user programs.

System Structure

★ MS-DOS

★ UNIX

★ OS/2 Layer Structure

★ Microkernel System Structure

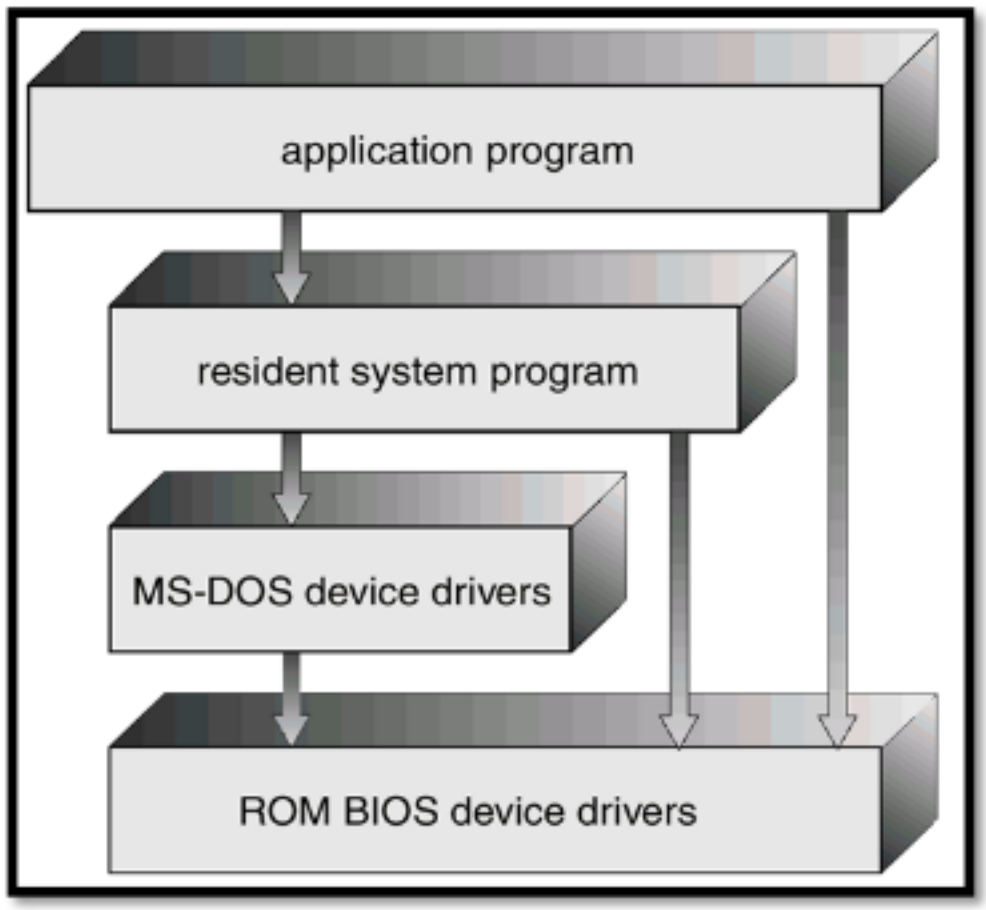
★ Windows NT Client-Server Structure

MS-DOS System Structure

★ MS-DOS – written to provide the most functionality in the least space

✦ not divided into modules.

✦ Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated.



UNIX System Structure

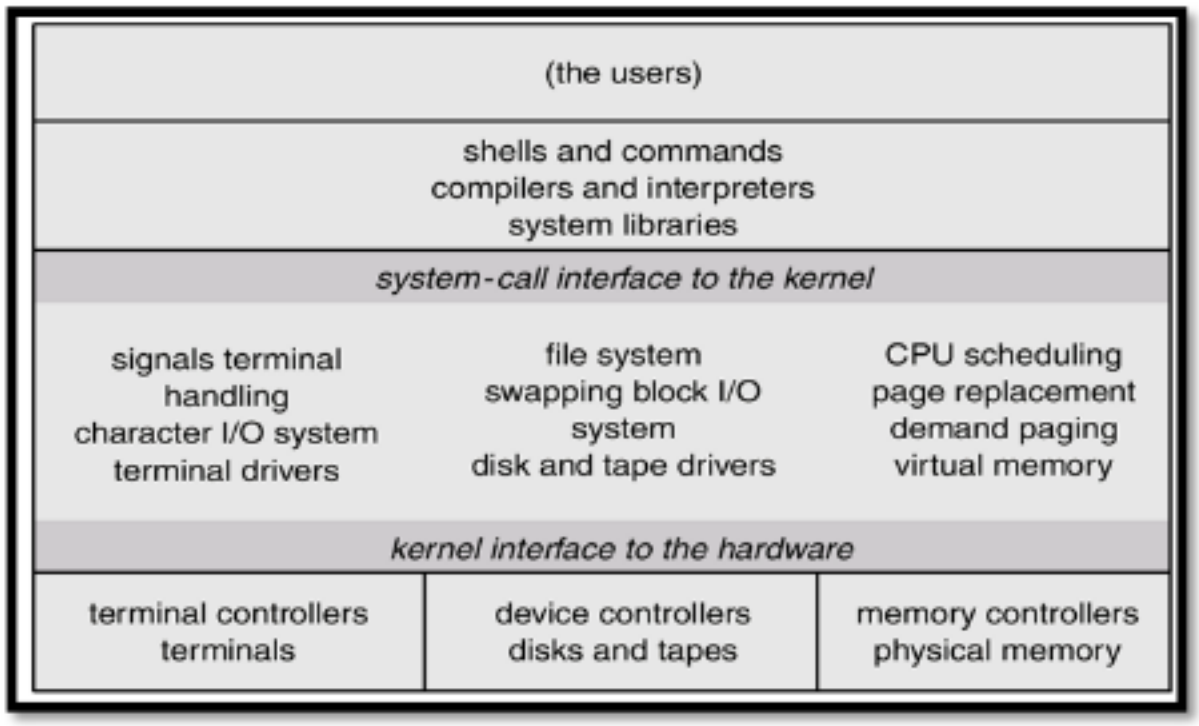
- UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring. The UNIX OS consists of two separable parts.

  ✦ 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.



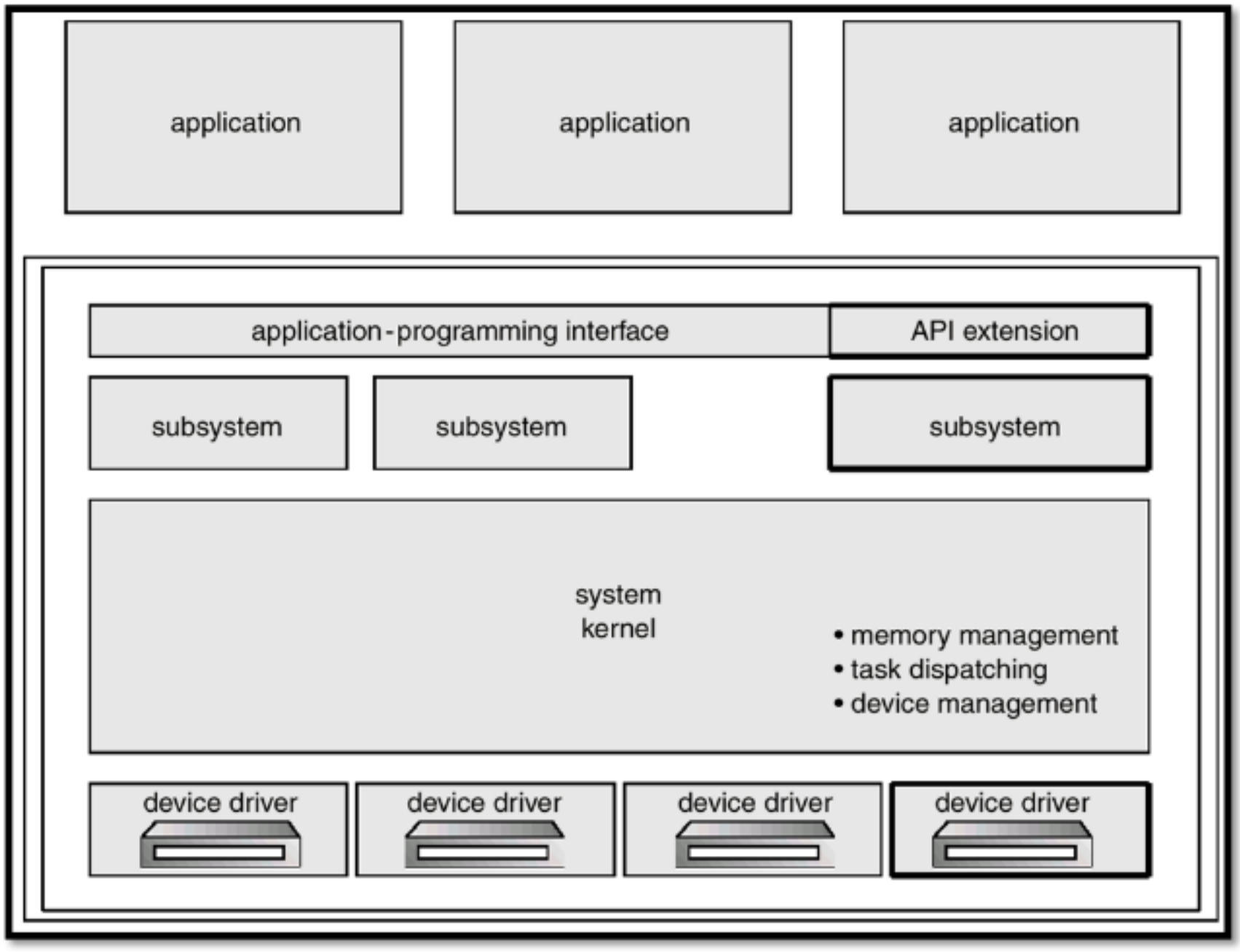
OS/2 Layer Structure

1. User Layer • Compatibility with OS/2, Windows 3.0 and DOS applications • App compatibility achieved using a Virtual DOS Machine (VDM) • VDM allowed the apps to run alongside another OS

2. Presentation Manager • The GUI of OS/2 • Supports mouse chording • Follows IBM's Common User Access (CUA) interface standard • Operates using a Single Input Queue (SIQ)

3. Issues with Single Input Queue • Blocked by a single unresponsive application • Caused the user interface to freeze • Never properly addressed and fixed

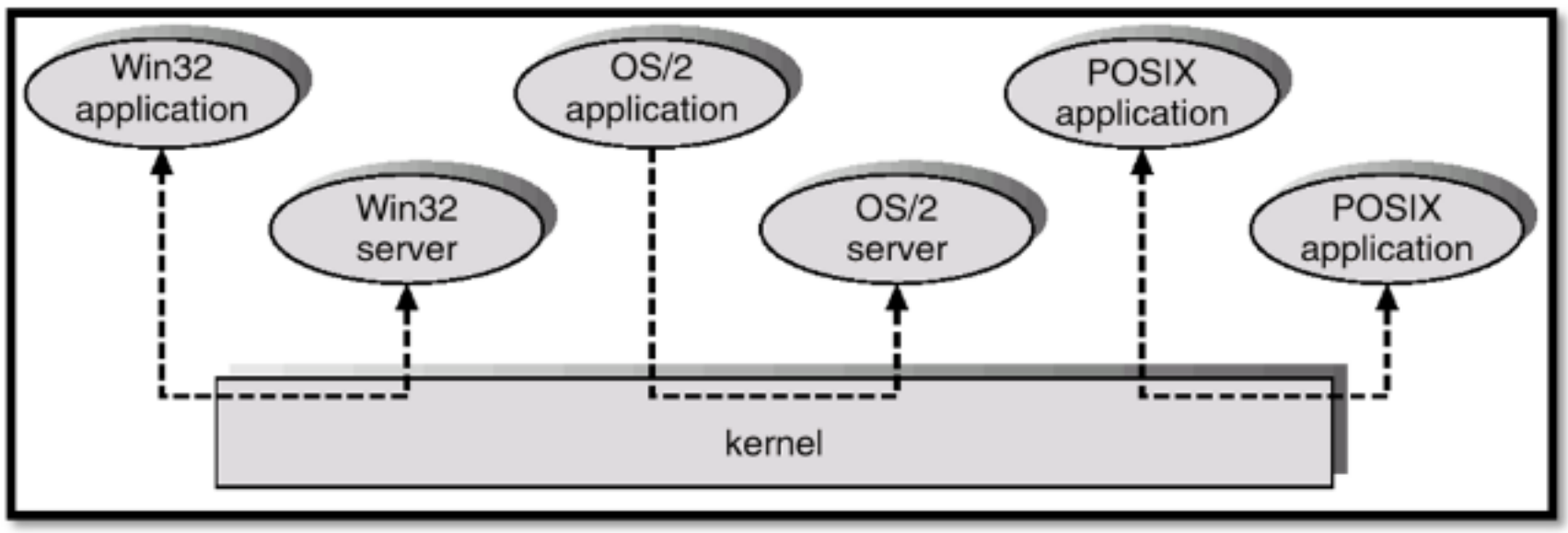
4. Kernel Layer • Boot manager handles improper boot procedures • Handles process switching by rewriting and reallocating memory • Memory is allocated in 64Kb pieces • Virtual device drivers exist outside the kernel • Each shutdown iteration occurs in two parts



Microkernel 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

Windows NT Client-Server Structure



Unit – II

WHAT IS PROCESS?

- "To introduce the notion of a process – a program in execution, which forms the basis of all computation

- To describe the various features of processes, including scheduling, creation and termination, and communication

- To explore inter process communication using shared memory and message Passing

- To describe communication in client-server systems

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
  - Program is ***passive*** entity stored on disk (executable file), process is ***active***
- Program becomes process when executable file loaded into memory
  - Execution of program started via GUI mouse clicks, command line entry of its name, etc.
  - One program can be several processes
- Consider multiple users executing the same program
  - Process has multiple parts

- The program code, also called text section
- Current activity including program counter, processor registers
- Stack containing temporary data

 Function parameters return addresses, local variables

Data section containing global variables

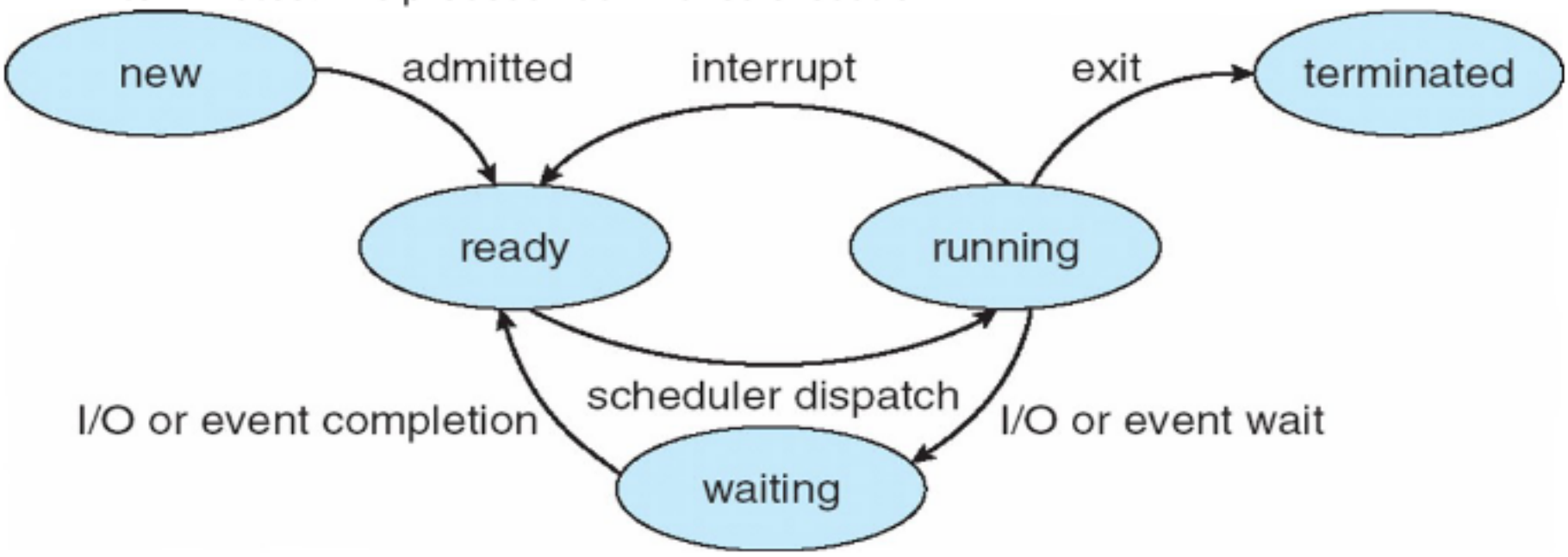
Heap containing memory dynamically allocated during run time

PROCESS STATE

AS A PROCESS EXECUTES, IT CHANGES STATE

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

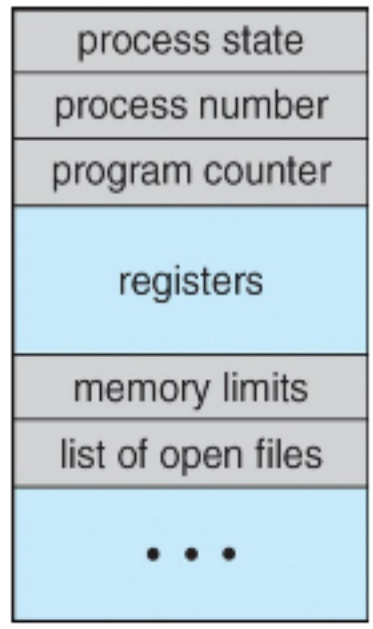


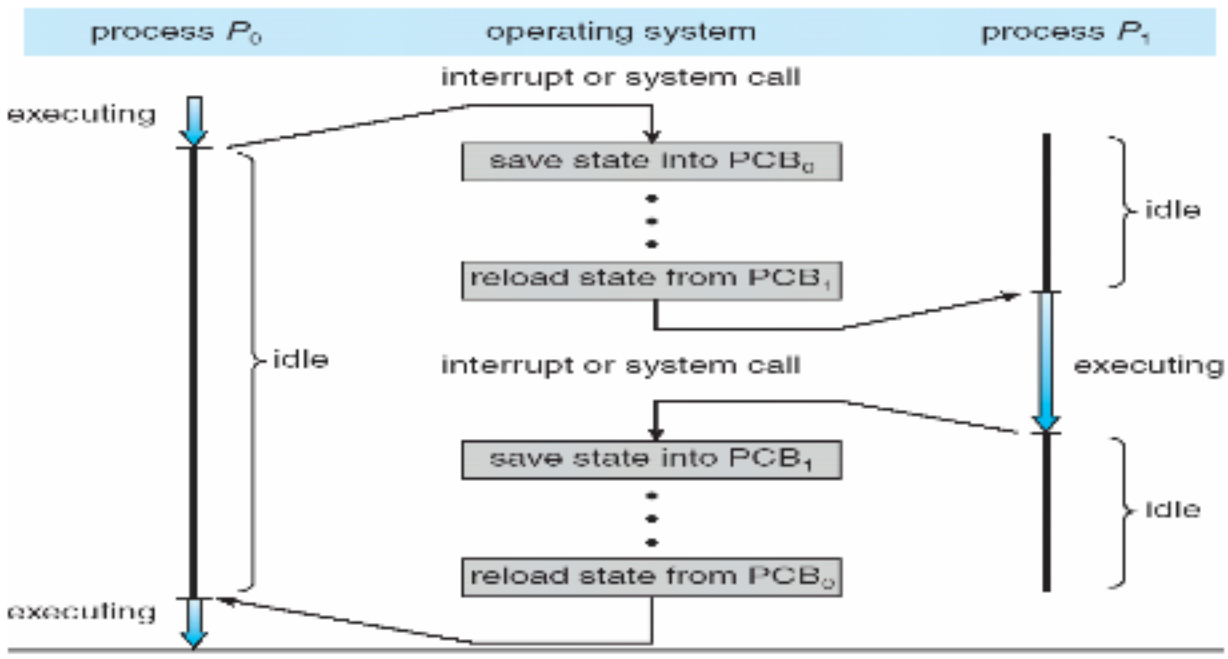
PROCESS CONTROL BLOCK

Information associated with each process (also called task control

block)

- Process state – running, waiting, etc.
- Program counter – location of instruction to next execute
- CPU registers – contents of all process-centric registers
- CPU scheduling information – priorities, scheduling queue pointers
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files





CPU SWITCH FROM PROCESS TO PROCESS

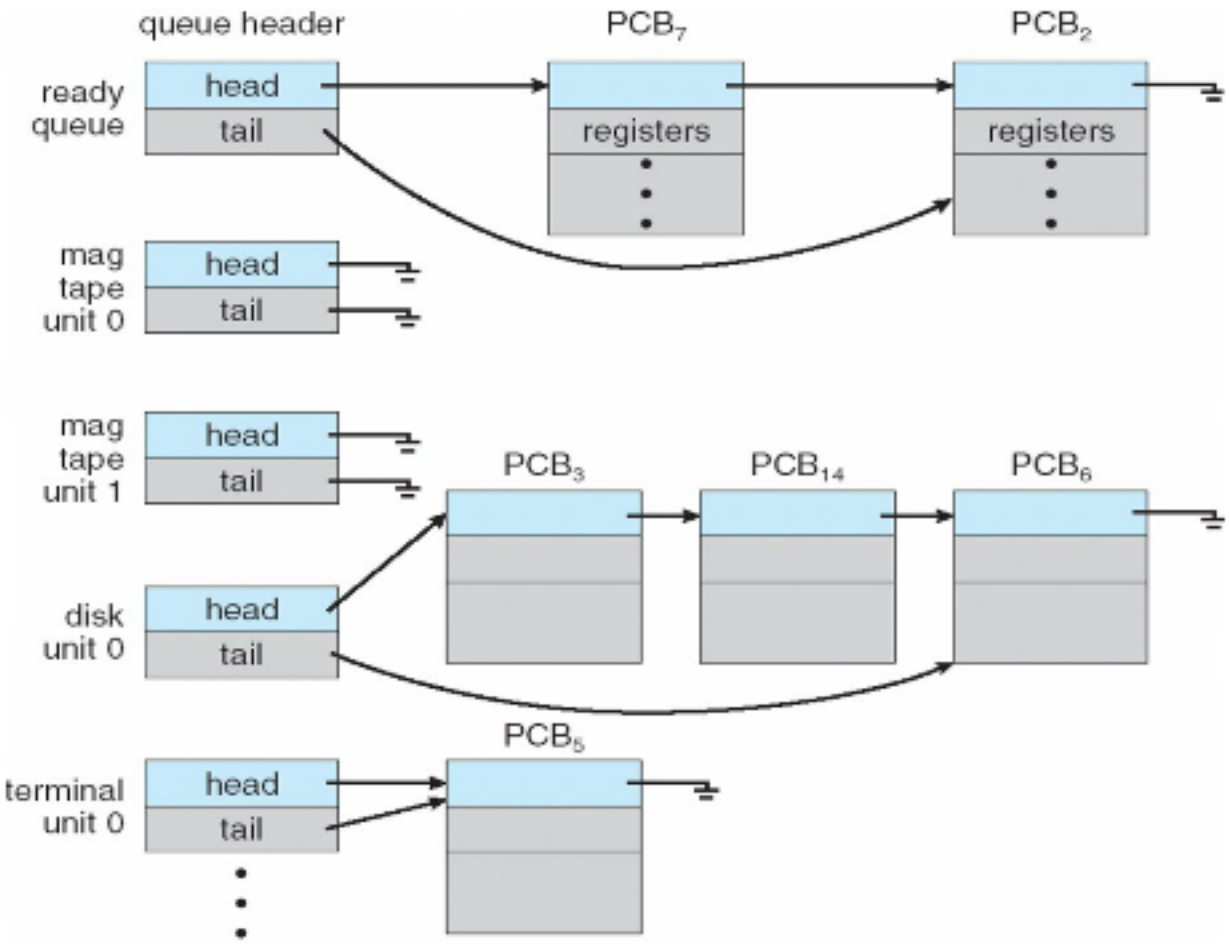
THREADS

- So far, process has a single thread of execution
- Consider having multiple program counters per process
  - Multiple locations can execute at once
    - Multiple threads of control -> threads
- Must then have storage for thread details, multiple program counters in PCB

PROCESS SCHEDULING

- Multiprogramming maximizes CPU use
- Time sharing quickly switches processes onto CPU to offer reactive CPU to all users
- Process scheduler selects among available processes for next execution on CPU
- Maintains scheduling queues of processes
  - 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



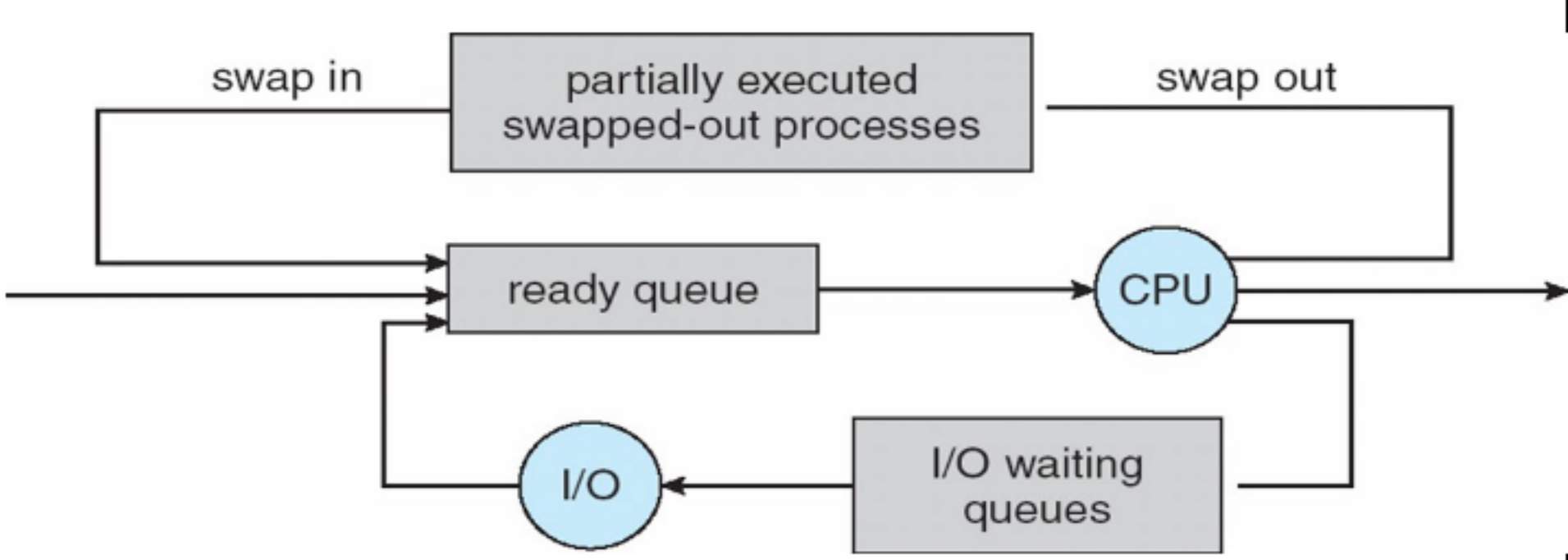
SCHEDULARS

- Long-term scheduler (or job scheduler) – selects which processes should be brought into the ready queue (e.g., often from a spool on disk)
- Short-term scheduler (or CPU scheduler) – selects which process should be executed next and allocates CPU
    - Sometimes the only scheduler in a system
  - Short-term scheduler is invoked very frequently (milliseconds) ⇒ (must be fast)
  - Long-term scheduler is invoked very infrequently (seconds, minutes) ⇒ (may be slow)
- The long-term scheduler controls the degree of multiprogramming (number of processes in memory)

- Processes can be described as either:
  - I/O-bound process – spends more time doing I/O than computations, many short CPU bursts
  - CPU-bound process – spends more time doing computations; few very long CPU bursts
- Long-term scheduler strives for good ***process mix***

ADDITION OF MEDIUM TERM SCHEDULAR

- Medium-term scheduler can be added if degree of multiple programming needs to decrease
  - Remove process from memory, store on disk, bring back in from disk to continue execution: swapping



MULTITASKING IN MOBILE SYSTEM

- Some systems / early systems allow only one process to run, others are suspended
- Due to screen real estate, user interface limits iOS to provide for a
  - Single foreground process – controlled via user interface

- Multiple background processes – in memory, running, but not on the display, and with limits
- Limits include single, short task, receiving notification of events, specific long-running tasks like audio playback

- Android runs foreground and background, with fewer limits

- Background process uses a service to perform tasks
  - Service can keep running even if background process is suspended
  - Service has no user interface, small memory use

CONTEXT SWITCH

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process via a context switch
  - Context of a process represented in the PCB
  - Context-switch time is overhead; the system does no useful work while switching
    - The more complex the OS and the PCB -> longer the context switch
  - Time dependent on hardware support
    - Some hardware provides multiple sets of registers per CPU -> multiple contexts loaded at once

OPERATIONS ON PROCESSES

The processes in the system can execute concurrently, and they must be created and deleted dynamically. Thus, the operating system must provide a mechanism (or facility) for process creation and termination.

Process Creation

A process may create several new processes, via a **create-process** system call, during the course of execution.

The creating process is called a **parent** process, whereas the new processes are called the **children** of that process.

Each of these new processes may in turn create other processes.

In general, a process will need certain resources (such as CPU time, memory, files, I/O devices) to accomplish its task.

When a process creates a subprocess, that subprocess may be able to obtain its resources directly from the operating system, or it may be constrained to a subset of the resources of the parent process.

The parent may have to partition its resources among its children, or it may be able to share some resources (such as memory or files) among several of its children.

Restricting a child process to a subset of the parent's resources prevents any process from overloading the system by creating too many subprocesses.

When a process is created it obtains, in addition to the various physical and logical resources, initialization data (or input) that may be passed along from the parent process to the child process.

For example, consider a process whose function is to display the status of a file, say F1, on the screen of a terminal.

When it is created, it will get, as an input from its parent process, the name of the file F1, and it will execute using that datum to obtain the desired information.

It may also get the name of the output device. Some operating systems pass resources to child processes.

On such a system, the new process may get two open files, F1 and the terminal device, and may just need to transfer the datum between the two.

When a process creates a new process, two possibilities exist in terms of execution:

1. The parent continues to execute concurrently with its children.
2. The parent waits until some or all of its children have terminated.

PROCESS TERMINATION

A process terminates when it finishes executing its final statement and asks the operating system to delete it by using the **exit** system call.

At that point, the process may return data (output) to its parent process (via the **wait** system call).

All the resources of the process-including physical and virtual memory, open files, and I/O buffers-are deallocated by the operating system.

Termination occurs under additional circumstances.

A process can cause the termination of another process via an appropriate system call (for example, **abort).**

Usually, only the parent of the process that is to be terminated can invoke such a system call. Otherwise, users could arbitrarily kill each other's jobs.

A parent therefore needs to know the identities of its children. Thus, when one process creates a new process, the identity of the newly created process is passed to the parent.

A parent may terminate the execution of one of its children for a variety of reasons, such as these:

The child has exceeded its usage of some of the resources that it has been allocated.

This requires the parent to have a mechanism to inspect the state of its children.

CPU SCHEDULING INTRODUCTION

Long-term scheduler is invoked very infrequently (seconds, minutes) (may be slow)

The long-term scheduler controls the *degree of multiprogramming (how many jobs are admitted to run on CPU)*

Short-term scheduler is invoked very frequently (milliseconds) (must be

fast) = “process scheduling on CPU”

Processes can be described as either:

- I/O-bound process – spends more time doing I/O than computations, many short CPU bursts
- CPU-bound process – spends more time
- doing computations; few very long CPU bursts

BASIC CONCEPTS

Maximum 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 (i.e. Ready Queue) that are ready to execute, and allocates the CPU to one of them.

1. Preemptive: allows a process to be interrupted in the midst of its CPU execution, taking the CPU away to another process
2. Non- Preemptive: ensures that a process relinquishes control of CPU when it finishes with its current CPU burst

CPU 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.

Preemptive: allows a process to be interrupted

Non- Preemptive: allows a process finishes with its current CPU burst

Scheduling under 1 and 4 is *non-preemptive*.

All other scheduling is *preemptive*.

SCHEDULING ALGORITHMS:

There are various algorithms which are used by the Operating System to schedule the processes on the processor in an efficient way.

The Purpose of a Scheduling algorithm

1. Maximum CPU utilization
2. Fare allocation of CPU
3. Maximum throughput
4. Minimum turnaround time
5. Minimum waiting time
6. Minimum response time

Scheduling types:

- First-Come, First-Served (FCFS) **Scheduling**.
- Shortest-Job-Next (SJN) **Scheduling**.
- Priority **Scheduling**.
- Shortest Remaining Time.
- Round Robin(RR) **Scheduling**.
- Multiple-Level Queues **Scheduling**.

FCFS Scheduling

First come first serve (FCFS) scheduling algorithm simply schedules the jobs according to their arrival time.

The job which comes first in the ready queue will get the CPU first. The lesser the arrival time of the job, the sooner will the job get the CPU.

FCFS scheduling may cause the problem of starvation if the burst time of the first process is the longest among all the jobs.

Advantages of FCFS

- Simple
- Easy
- First come, First serve

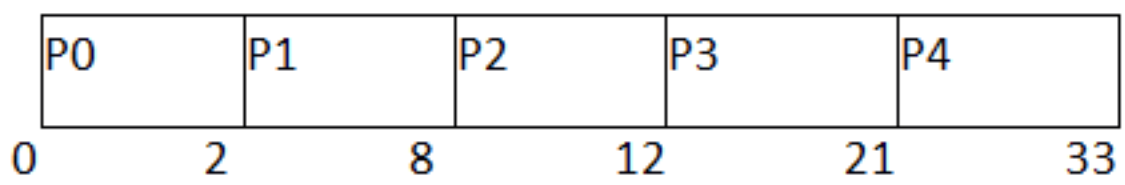
Disadvantages of FCFS

1. The scheduling method is non preemptive, the process will run to the completion.
2. Due to the non-preemptive nature of the algorithm, the problem of starvation may occur.
3. Although it is easy to implement, but it is poor in performance since the average waiting time is higher as compare to other scheduling algorithms.

Example

Let's take an example of The FCFS scheduling algorithm. In the Following schedule, there are 5 processes with process ID P0, P1, P2, P3 and P4. P0 arrives at time 0, P1 at time 1, P2 at time 2, P3 arrives at time 3 and Process P4 arrives at time 4 in the ready queue. The processes and their respective Arrival and Burst time are given in the following table.

The Turnaround time and the waiting time are calculated by using the following formula.

1. Turn Around Time = Completion Time - Arrival Time
2. Waiting Time = Turnaround time - Burst Time
3. Avg Waiting Time=31/5
4. 

5. (Gantt chart)

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule is:

- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

  Suppose that the processes arrive in the order:

  P_2 , P_3 , P_1

- The Gantt chart for the schedule is:
- Waiting time for P_1 = 6; P_2 = 0; P_3 = 3
- Average waiting time: (6 + 0 + 3)/3 = 3
- Much better than previous case
- Convoy effect - short process behind long process
  - Consider one CPU-bound and many I/O-bound processes

Shortest Job First (SJF) Scheduling

Till now, we were scheduling the processes according to their arrival time (in FCFS scheduling). However, SJF scheduling algorithm, schedules the processes according to their burst time.

In SJF scheduling, the process with the lowest burst time, among the list of available processes in the ready queue, is going to be scheduled next.

However, it is very difficult to predict the burst time needed for a process hence this algorithm is very difficult to implement in the system.

Advantages of SJF

1. Maximum throughput
2. Minimum average waiting and turnaround time

Disadvantages of SJF

1. May suffer with the problem of starvation

2. It is not implementable because the exact Burst time for a process can't be known in advance.

There are different techniques available by which, the CPU burst time of the process can be determined. We will discuss them later in detail.

Example

In the following example, there are five jobs named as P1, P2, P3, P4 and P5. Their arrival time and burst time are given in the table below.

- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time
- SJF is optimal – gives minimum average waiting time for a given set of processes
  - The difficulty is knowing the length of the next CPU request
  - Could ask the user

Example of SJF

Process	Arriva l Time	Burst Time
P_1	0.0	6
P_2	2.0	8
P_3	4.0	7
P_4	5.0	3

- SJF scheduling chart

  Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

Priority Based Scheduling

- Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems.
- Each process is assigned a priority. Process with highest priority is to be executed first and so on.
- Processes with same priority are executed on first come first served

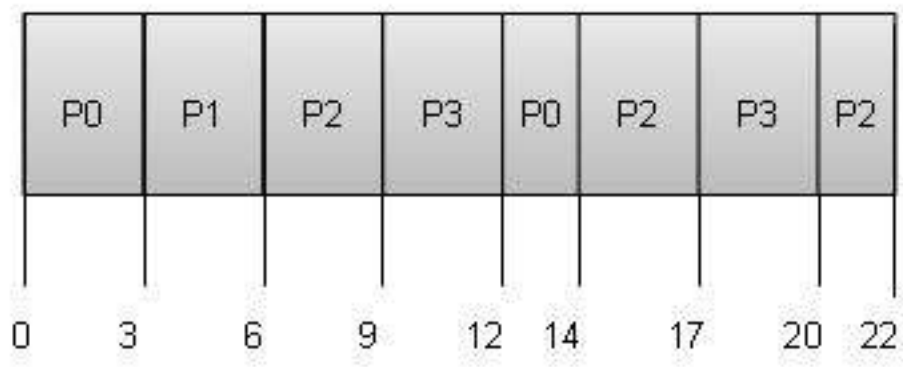
basis.

- Priority can be decided based on memory requirements, time requirements or any Average Wait Time: (0 + 10 + 12 + 2)/4 = 24 / 4 = 6

Round Robin Scheduling

- Round Robin is the preemptive process scheduling algorithm.
- Each process is provided a fix time to execute, it is called a **quantum**.
- Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
- Context switching is used to save states of preempted processes.

Quantum = 3



Wait time of each process is as follows –

Process	Wait Time : Service Time - Arrival Time
P0	(0 - 0) + (12 - 3) = 9
P1	(3 - 1) = 2
P2	(6 - 2) + (14 - 9) + (20 - 17) = 12
P3	(9 - 3) + (17 - 12) = 11

Average Wait Time: (9+2+12+11) / 4 = 8.5

Multiple-Level Queues Scheduling

Multiple-level queues are not an independent scheduling algorithm. They make

use of other existing algorithms to group and schedule jobs with common characteristics.

- Multiple queues are maintained for processes with common characteristics.
- Each queue can have its own scheduling algorithms.
- Priorities are assigned to each queue.

For example, CPU-bound jobs can be scheduled in one queue and all I/O-bound jobs in another queue. The Process Scheduler then alternately selects jobs from each queue and assigns them to the CPU based on the algorithm assigned to the queue.

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
  - o Currently, most common
- ★ Processor affinity – process has affinity for processor on which it is currently running
  - o soft affinity
  - ★ hard affinity
  - ★ Variations including processor sets

Real-Time CPU Scheduling

- ★ Can present obvious challenges
- ★ Soft real-time systems – no guarantee as to when critical real-time process will be scheduled
- ★ Hard real-time systems – task must be serviced by its deadline
- ★ Two types of latencies affect performance
  1. Interrupt latency – time from arrival of interrupt to start of routine that services interrupt
  2. Dispatch latency – time for schedule to take current process off CPU and switch to another

UNIT-III

Process Synchronization

Process Synchronization means sharing system resources by processes in a such a way that, Concurrent access to shared data is handled thereby minimizing the chance of inconsistent data.

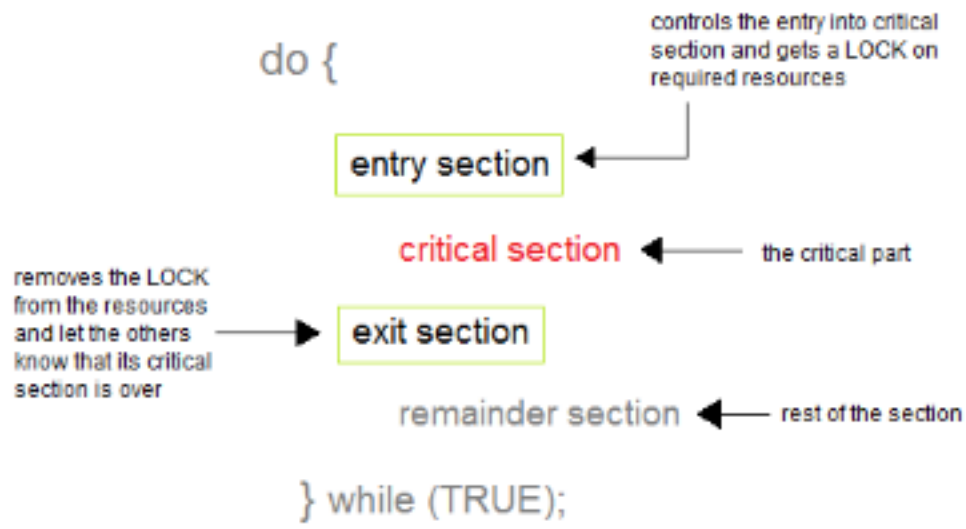
Maintaining data consistency demands mechanisms to ensure synchronized execution of cooperating processes.

Process Synchronization was introduced to handle problems that arose while multiple process executions. Some of the problems are discussed below.

Critical Section Problem

A Critical Section is a code segment that accesses shared variables and has to be executed as an atomic action. It means that in a group of cooperating processes, at a given point of time, only one process must be executing its critical section.

If any other process also wants to execute its critical section, it must wait until the first one finishes.



SOLUTION TO CRITICAL SECTION PROBLEM

A solution to the critical section problem must satisfy the following three conditions:

1. Mutual Exclusion

Out of a group of cooperating processes, only one process can be in its critical section at a given point of time.

2. Progress

If no process is in its critical section, and if one or more threads want to execute their critical section then any one of these threads must be allowed to get into its critical section.

3. Bounded Waiting

After a process makes a request for getting into its critical section, there is a limit for how many other processes can get into their critical section, before this process's request is granted. So after the limit is reached, system must grant the process permission to get into its critical section.

SYNCHRONIZATION HARDWARE

Many systems provide hardware support for critical section code.

The critical section problem could be solved easily in a single-processor environment if we could disallow interrupts to occur while a shared variable or resource is being modified.

In this manner, we could be sure that the current sequence of instructions would be allowed to execute in order without pre-emption.

Unfortunately, this solution is not feasible in a multiprocessor environment.

Disabling interrupt on a multiprocessor environment can be time consuming as the message is passed to all the processors.

This message transmission lag, delays entry of threads into critical section and the system efficiency decreases.

MUTEX LOCKS

As the synchronization hardware solution is not easy to implement for everyone, a strict software approach called Mutex Locks was introduced.

In this approach, in the entry section of code, a LOCK is acquired over the critical resources modified and used inside critical section, and in the exit section that LOCK is released.

As the resource is locked while a process executes its critical section hence no other process can access it.

INTRODUCTION TO SEMAPHORES

In 1965, Dijkstra proposed a new and very significant technique for managing concurrent processes by using the value of a simple integer variable to synchronize the progress of interacting processes. This integer variable is called **semaphore**.

So it is basically a synchronizing tool and is accessed only through two low standard atomic operations, **wait** and **signal** designated by P(S) and V(S) respectively.

In very simple words, **semaphore** is a variable which can hold only a non-negative Integer value, shared between all the threads, with operations **wait** and **signal**, which work as follow:

```
P(S): if S ≥ 1 then S := S - 1
      else <block and enqueue the process>;

V(S): if <some process is blocked on the queue>
        then <unblock a process>
      else S := S + 1;
```

The classical definitions of **wait** and **signal** are:

- **Wait**: Decrements the value of its argument S, as soon as it would become non-negative(greater than or equal to 1).
- **Signal**: Increments the value of its argument S, as there is no more process

blocked on the queue.

PROPERTIES OF SEMAPHORES

1. It's simple and always have a non-negative Integer value.
2. Works with many processes.
3. Can have many different critical sections with different semaphores.
4. Each critical section has unique access semaphores.
5. Can permit multiple processes into the critical section at once, if desirable.

TYPES OF SEMAPHORES

Semaphores are mainly of two types:

1. **Binary Semaphore:**

It is a special form of semaphore used for implementing mutual exclusion, hence it is often called a **Mutex**. A binary semaphore is initialized to 1 and only takes the values 0 and 1 during execution of a program.

2. **Counting Semaphores:**

   These are used to implement bounded concurrency.

Example of Use

Here is a simple step wise implementation involving declaration and usage of semaphore.

```
Shared var mutex: semaphore = 1;
Process i
   begin
   .
   .
   P(mutex);
   execute CS;
   V(mutex);
   .
   .
   End;
```

LIMITATIONS OF SEMAPHORES

1. **Priority Inversion** is a big limitation of semaphores.
2. Their use is not enforced, but is by convention only.
3. With improper use, a process may block indefinitely. Such a situation is called **Deadlock**. We will be studying deadlocks in details in coming lessons.

CLASSICAL PROBLEMS OF SYNCHRONIZATION

In this tutorial we will discuss about various classic problem of synchronization.

Semaphore can be used in other synchronization problems besides Mutual Exclusion.

Below are some of the classical problem depicting flaws of process synchronaization in systems where cooperating processes are present.

We will discuss the following three problems:

1. Bounded Buffer (Producer-Consumer) Problem
2. The Readers Writers Problem
3. Dining Philosophers Problem

BOUNDED BUFFER PROBLEM

- This problem is generalized in terms of the **Producer Consumer problem**, where a **finite** buffer pool is used to exchange messages between producer and consumer processes.

Because the buffer pool has a maximum size, this problem is often called the **Bounded buffer problem**.

- Solution to this problem is, creating two counting semaphores "full" and "empty" to keep track of the current number of full and empty buffers respectively.

THE READERS WRITERS PROBLEM

- In this problem there are some processes(called **readers**) that only read the shared data, and never change it, and there are other processes(called **writers**) who may change the data in addition to reading, or instead of reading it.

- There are various type of readers-writers problem, most centred on relative priorities of readers and writers.

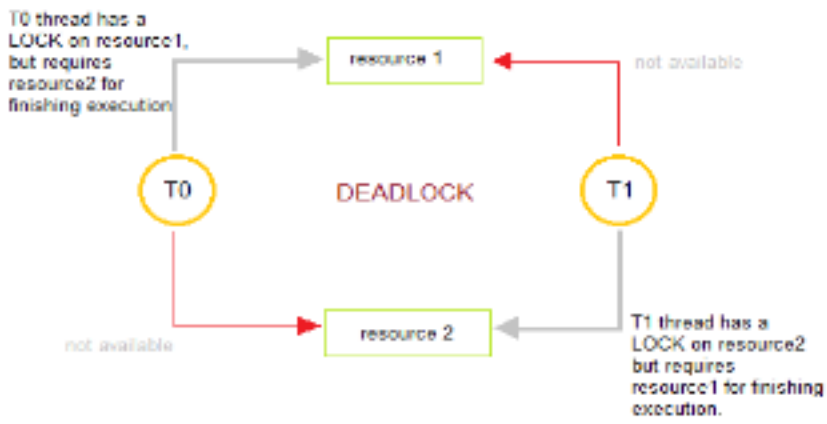
DINING PHILOSOPHERS PROBLEM

- The dining philosopher's problem involves the allocation of limited resources to a group of processes in a deadlock-free and starvation-free manner.
- There are five philosophers sitting around a table, in which there are five chopsticks/forks kept beside them and a bowl of rice in the centre, When a philosopher wants to eat, he uses two chopsticks - one from their left and one from their right. When a philosopher wants to think, he keeps down both chopsticks at their original place.

DEAD LOCKS

What is a Deadlock?

Deadlocks are a set of blocked processes each holding a resource and waiting to acquire a resource held by another process.



Example:

A common example is the traffic deadlock. Figure 1 shows a situation in which four cars have arrived at a four-way stop intersection at approximately the same time.

The four quadrants of the intersection are the resources over which control is needed. In particular, if all four cars wish to go straight through the intersection, the resource requirements are as follows:

- Car 1, traveling north, needs quadrants a and b.
- Car 2 needs quadrants b and c.

Figure 1 Illustration of Deadlock

- Car 3 needs quadrants c and d.
- Car 4 needs quadrants d and a.

SYSTEM MODEL

A system model or structure consists of a fixed number of resources to be circulated among some opposing processes.

The resources are then partitioned into numerous types, each consisting of some specific quantity of identical instances. Memory space, CPU cycles, directories and files, I/O devices like keyboards, printers and CD-DVD drives are prime examples of resource types.

When a system has 2 CPUs, then the resource type CPU got two instances.

Under the standard mode of operation, any process may use a resource in only the below-mentioned sequence:

1. **REQUEST**: When the request can't be approved immediately (where the case may be when another process is utilizing the resource), then the requesting job must remain waited until it can obtain the resource.
2. **USE**: The process can run on the resource (like when the resource is a printer, its job/ process is to print on the printer).

- **RELEASE**: The process releases the resource (like, terminating or exiting any specific process).

DEADLOCK CHARACTERIZATION

A deadlock state can occur when the following four circumstances hold simultaneously within a system:

- **MUTUAL EXCLUSION:** At least there should be one resource that has to be held in a non-sharable manner; i.e., only a single process at a time can utilize the resource. If other process demands that resource, the requesting process must be postponed until the resource gets released.
- **HOLD AND WAIT:** A job must be holding at least one single resource and waiting to obtain supplementary resources which are currently being held by several other processes.
- **NO PREEMPTION**: Resources can't be anticipated; i.e., a resource can get released only willingly by the process holding it, then after that, the process has completed its task.
- **CIRCULAR WAIT:** The circular - wait situation implies the hold-and-wait state or condition, and hence all the four conditions are not completely independent. They are interconnected among each other.

DEADLOCK PREVENTION:

HANDLING DEADLOCK

The above points focus on preventing deadlocks. But what to do once a deadlock has occurred. Following three strategies can be used to remove deadlock after its occurrence.

1. **Preemption**

   We can take a resource from one process and give it to other. This will resolve the deadlock situation, but sometimes it does causes problems.

2. **Rollback**

   In situations where deadlock is a real possibility, the system can periodically make a record of the state of each process and when deadlock occurs, roll everything back to the last checkpoint, and restart, but allocating resources differently so that deadlock does not occur.

3. **Kill one or more processes**

   This is the simplest way, but it works.

Methods for Handling Deadlocks

Normally you can deal with the deadlock issues and situations in one of the three

ways mentioned below:

- You can employ a protocol for preventing or avoiding deadlocks, and ensure that the system will never go into a deadlock state.
- You can let the system to enter any deadlock condition, detect it, and then recover.
- You can overlook the issue altogether and assume that deadlocks never occur within the system.

DEADLOCK PREVENTION

An indirect method of deadlock prevention is to prevent the occurrence of one of the three necessary conditions prevented..

A direct method of deadlock prevention is to prevent the occurrence of a circular wait

(item 4). The techniques related to each of the four conditions.

Mutual Exclusion

If access to a resource requires mutual exclusion, then mutual exclusion must be supported by the OS. Some resources, such as files, may allow multiple accesses for reads but only exclusive access for writes. Deadlock can occur if more than one process requires write permission.

Hold and Wait

The hold-and-wait condition can be prevented by requiring that a process request all of its required resources at one time and blocking the process until all requests can be granted simultaneously. This approach is inefficient in two ways.

First, a process may be held up for a long time waiting for all of its resource requests to be filled.

Second, resources allocated to a process may remain unused for a considerable period, during which time they are denied to other processes.

No Preemption

This condition can be prevented in several ways. First, if a process holding certain resources is denied a further request, that process must release its original resources and, if necessary, request them again together with the additional resource.

If a process requests a resource that is currently held by another process, the OS may preempt the second process and require it to release its resources.

This approach is practical only when applied to resources whose state can be easily saved and restored later.

Circular Wait

The circular-wait condition can be prevented by defining a linear ordering of resource types. If a process has been allocated resources of type *R*, then it may subsequently request only those resources of types following *R* in the ordering.

DEADLOCK AVOIDANCE

In **deadlock prevention**, we constrain resource requests to prevent at least one of the four conditions of deadlock. One of the three necessary policy conditions (mutual exclusion, hold and wait, no preemption), or directly by preventing circular wait. **Deadlock avoidance**, on the other hand, allows the three necessary conditions but makes judicious choices to assure that the deadlock point is never reached.

Hence avoidance allows more concurrency than prevention. A decision is made dynamically whether the current resource allocation request will, if granted, potentially lead to a deadlock.

We describe two approaches to deadlock avoidance:

- Do not start a process if its demands might lead to deadlock.
- Do not grant an incremental resource request to a process if this allocation might lead to deadlock.

Process Initiation Denial

Consider a system of *n* processes and *m* different types of resources. Let us define the following vectors and matrices:

[Resource = $\mathbf{R} = (R_1, R_2, \ldots, R_m)$	total amount of each resource in the system
Available = $\mathbf{V} = (V_1, V_2, \ldots, V_m)$	total amount of each resource not allocated to any process
Claim = $\mathbf{C} = \begin{bmatrix} C_{11} & C_{12} & \cdots & C_{1m} \\ C_{21} & C_{22} & \cdots & C_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ C_{n1} & C_{n2} & \cdots & C_{nm} \end{bmatrix}$	C_{ij} = requirement of process i for resource j
Allocation = $\mathbf{A} = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1m} \\ A_{21} & A_{22} & \cdots & A_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nm} \end{bmatrix}$	A_{ij} = current allocation to process i of resource j

The matrix Claim gives the maximum requirement of each process for each resource, with one row dedicated to each process. This information must be declared in advance by a process for deadlock avoidance to work.

Resource Allocation

Banker's algorithm

Consider a system with a fixed number of processes and a fixed number of resources. At any time a process may have zero or more resources allocated to it. The **state** of the system reflects the current allocation of resources to processes.

A **safe state** is one in which there is at least one sequence of resource allocations to processes that does not result in a deadlock (i.e., all of the processes can be run to completion). An **unsafe state** is, of course, a state that is not safe.

The following example illustrates these concepts. Figure 6.7a shows the state of a system consisting of four processes and three resources. The total amount of resources R1, R2, and R3 are 9, 3, and 6 units, respectively. In the current state allocations have been made to the four processes, leaving 1 unit of R2 and 1 unit of R3 available.

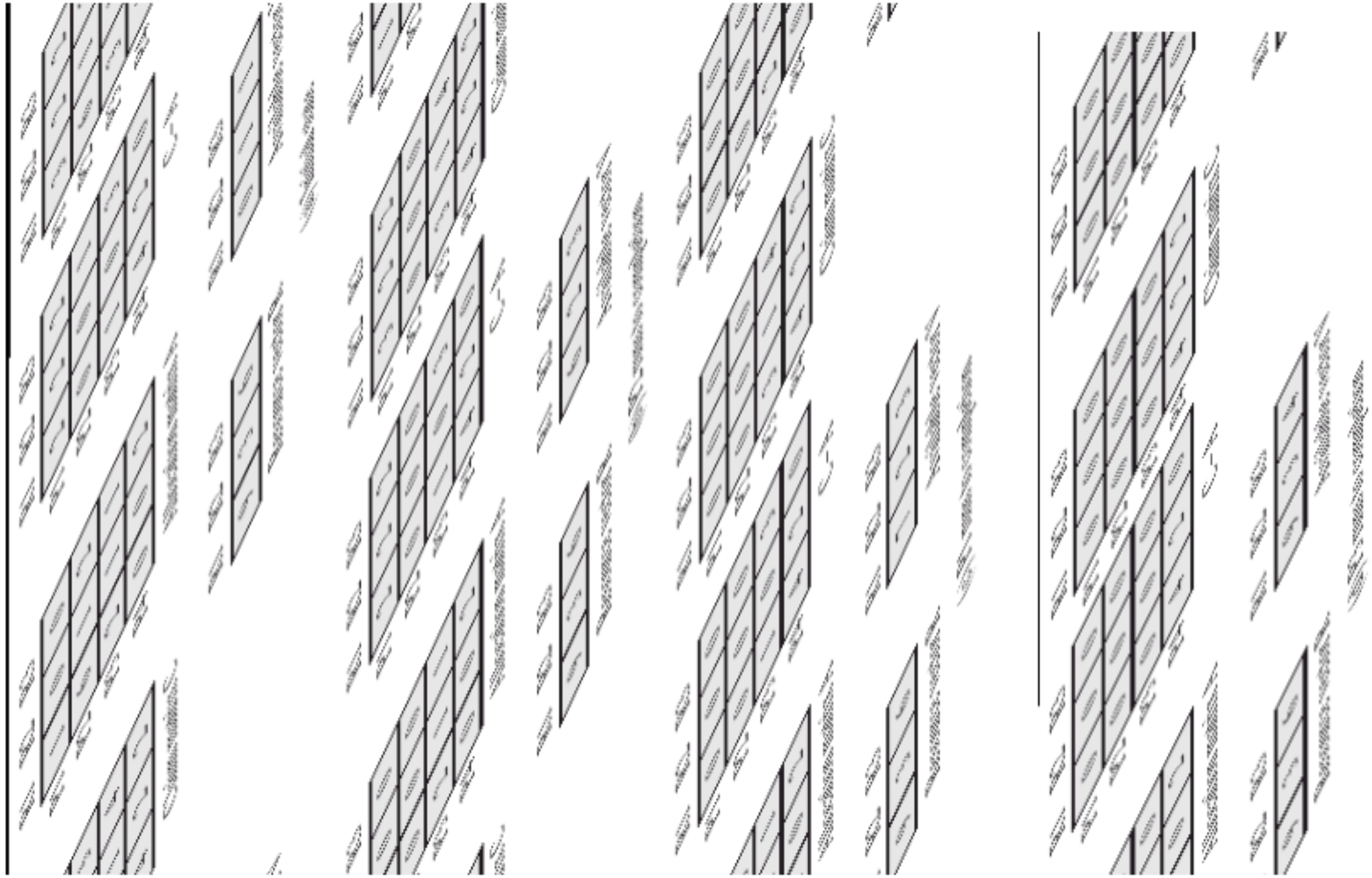


Figure 6.7 Determination of a Safe State

Figure 6.8 Determination of an Unsafe State

Consider the state defined in Figure 6.8a. Suppose P2 makes a request for one additional unit of R1 and one additional unit of R3. If we assume the request is granted, then the resulting state is that of Figure 6.7a.We have already seen that this is a safe state; therefore, it is safe to grant the request. Now let us return to the state of Figure 6.8a and suppose that P1 makes the request for one additional unit each of R1 and R3; if we assume that the request is granted, we are left in the state of Figure 6.8b.

Deadlock avoidance has the advantage that it is not necessary to preempt and rollback processes, as in deadlock detection, and is less restrictive than deadlock prevention. A number of restrictions on its use:

- The maximum resource requirement for each process must be stated in advance.
- The processes under consideration must be independent; that is, the order in which they execute must be unconstrained by any synchronization requirements.

• There must be a fixed number of resources to allocate.

• No process may exit while holding resources.

DEADLOCK DETECTION

Deadlock Detection Algorithm

Each resource request has two advantages: it leads to early detection, and the algorithm is relatively simple because it is based on incremental changes to the state of the system. On the other hand, such frequent checks consume considerable processor time.

We can use Figure 6.10 to illustrate the deadlock detection algorithm. The algorithm proceeds as follows:

1. Mark P4, because P4 has no allocated resources.

2. Set **W =** (0 0 0 0 1).

3. The request of process P3 is less than or equal to **W**, so mark P3 and set

W=W + (0 0 0 1 0) = (0 0 0 1 1).

4. No other unmarked process has a row in **Q** that is less than or equal to **W**. Therefore, terminate the algorithm.

The algorithm concludes with P1 and P2 unmarked, indicating that these processes are deadlocked.

Figure 6.10 Example for Deadlock Detection

Recovery

Once deadlock has been detected, some strategy is needed for recovery.The following are possible approaches, listed in order of increasing sophistication:

1. Abort all deadlocked processes.

2. Back up each deadlocked process to some previously defined checkpoint, and restart all processes. This requires that rollback and restart mechanisms be built in to the system.

3. Successively abort deadlocked processes until deadlock no longer exists. The order in which processes are selected for abortion should be on the basis of some criterion of minimum cost.

4. Successively preempt resources until deadlock no longer exists.

For (3) and (4), the selection criteria could be one of the following. Choose the process with the

• least amount of processor time consumed so far

• least amount of output produced so far

• most estimated time remaining

• least total resources allocated so far

• lowest priority

Some of these quantities are easier to measure than others. Estimated time remaining is particularly suspected.

UNIT-IV

MEMEORY MANAGEMENT INTRODUCTION:

Memory management is the functionality of an operating system which handles or manages primary memory and moves processes back and forth between main memory and disk during execution.

Memory management keeps track of each and every memory location, regardless of either it is allocated to some process or it is free.

It checks how much memory is to be allocated to processes. It decides which process will get memory at what time. It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.

This concept will teach you basic concepts related to Memory Management.

PROCESS ADDRESS SPACE

The process address space is the set of logical addresses that a process references in its code. For example, when 32-bit addressing is in use, addresses can range from 0 to 0x7fffffff; that is, 2^31 possible numbers, for a total theoretical size of 2 gigabytes.

The operating system takes care of mapping the logical addresses to physical addresses at the time of memory allocation to the program. There are three types of addresses used in a program before and after memory is allocated −

S.N.	Memory Addresses & Description
1	**Symbolic addresses** The addresses used in a source code. The variable names, constants, and instruction labels are the basic elements of the symbolic address space.
2	**Relative addresses** At the time of compilation, a compiler converts symbolic addresses into relative addresses.
3	**Physical addresses** The loader generates these addresses at the time when a program is loaded

	into main memory.

Virtual and physical addresses are the same in compile-time and load-time address-binding schemes. Virtual and physical addresses differ in execution-time address-binding scheme.

The set of all logical addresses generated by a program is referred to as a **logical address space**. The set of all physical addresses corresponding to these logical addresses is referred to as a **physical address space.**

The runtime mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device.

MMU uses following mechanism to convert virtual address to physical address.

- The value in the base register is added to every address generated by a user process, which is treated as offset at the time it is sent to memory. For example, if the base register value is 10000, then an attempt by the user to use address location 100 will be dynamically reallocated to location 10100.
- The user program deals with virtual addresses; it never sees the real physical addresses.

STATIC VS DYNAMIC LOADING

The choice between Static or Dynamic Loading is to be made at the time of computer program being developed.

If you have to load your program statically, then at the time of compilation, the complete programs will be compiled and linked without leaving any external program or module dependency.

The linker combines the object program with other necessary object modules into an absolute program, which also includes logical addresses.

If you are writing a Dynamically loaded program, then your compiler will compile the program and for all the modules which you want to include dynamically, only references will be provided and rest of the work will be done at the time of execution.

At the time of loading, with **static loading**, the absolute program (and data) is loaded into memory in order for execution to start.

If you are using **dynamic loading**, dynamic routines of the library are stored on a disk in relocatable form and are loaded into memory only when they are needed by the program.

STATIC VS DYNAMIC LINKING

As explained above, when static linking is used, the linker combines all other modules needed by a program into a single executable program to avoid any runtime dependency.

When dynamic linking is used, it is not required to link the actual module or library with the program, rather a reference to the dynamic module is provided at the time of compilation and linking. Dynamic Link Libraries (DLL) in Windows and Shared Objects in Unix are good examples of dynamic libraries.

SWAPPING

SWAPPING INTRODUCTION

Swapping is a mechanism in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes.

At some later time, the system swaps back the process from the secondary storage to main memory.

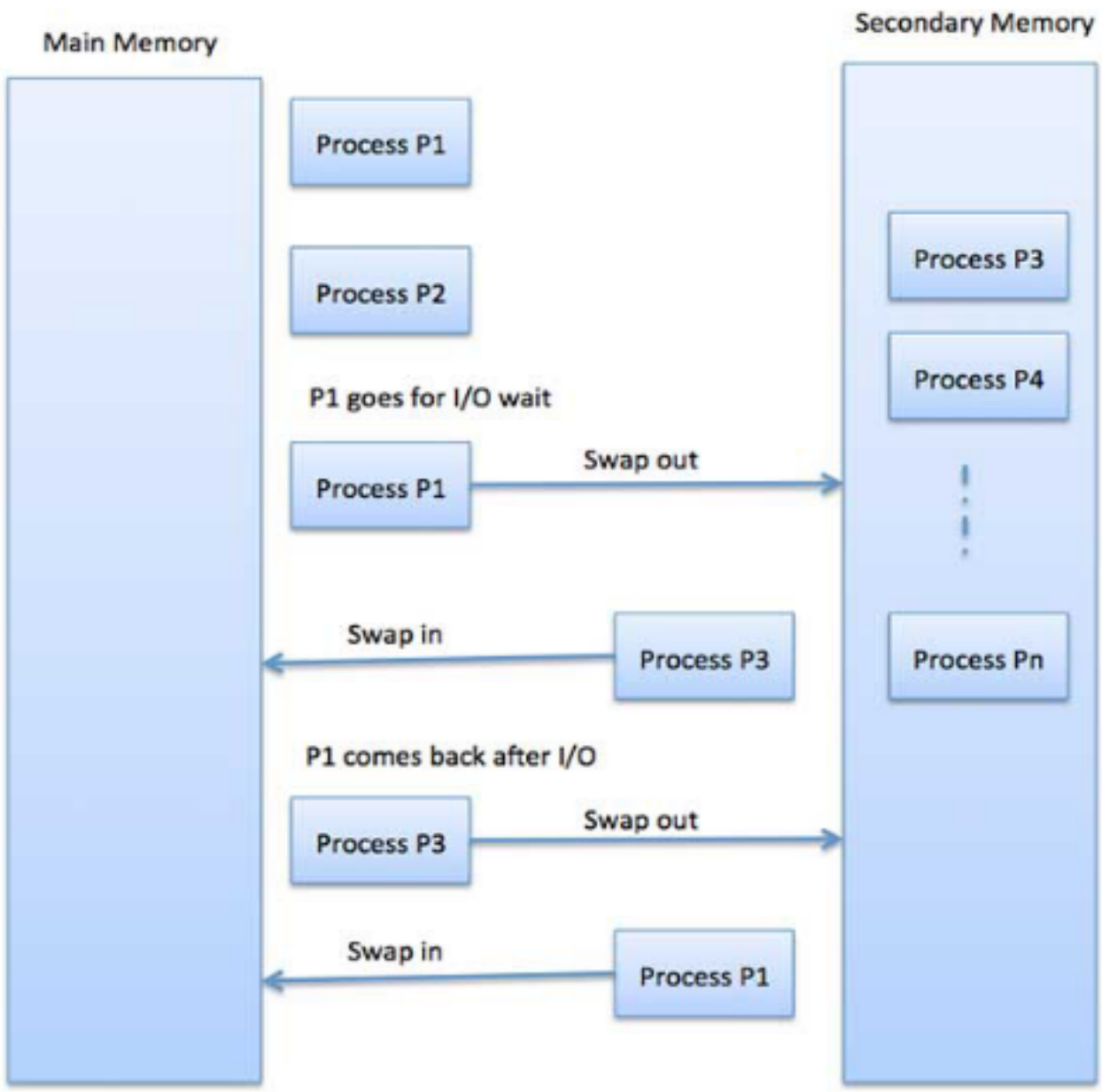
Though performance is usually affected by swapping process but it helps in running multiple and big processes in parallel and that's the reason **Swapping is also known as a technique for memory compaction**.

The total time taken by swapping process includes the time it takes to move the entire process to a secondary disk and then to copy the process back to memory, as well as the time the process takes to regain main memory.

Let us assume that the user process is of size 2048KB and on a standard hard disk where swapping will take place has a data transfer rate around 1 MB per second. The actual transfer of the 1000K process to or from memory will take

```
2048KB / 1024KB per second
= 2 seconds
= 2000 milliseconds
```

Now considering in and out time, it will take complete 4000 milliseconds plus other overhead where the process competes to regain main memory.



MEMORY ALLOCATION

MEMORY ALLOCATION INTRODUCTION

Main memory usually has two partitions −

- **Low Memory** − Operating system resides in this memory.
- **High Memory** − User processes are held in high memory.

Operating system uses the following memory allocation mechanism.

S.N.	Memory Allocation & Description
1	**Single-partition allocation** In this type of allocation, relocation-register scheme is used to protect user processes from each other, and from changing operating-system code and data. Relocation register contains value of smallest physical address whereas limit register contains range of logical addresses. Each logical address must be less than the limit register.
2	**Multiple-partition allocation**

	In this type of allocation, main memory is divided into a number of fixed-sized partitions where each partition should contain only one process. When a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process.

FRAGMENTATION

FRAGMENTATION

As processes are loaded and removed from memory, the free memory space is broken into little pieces.

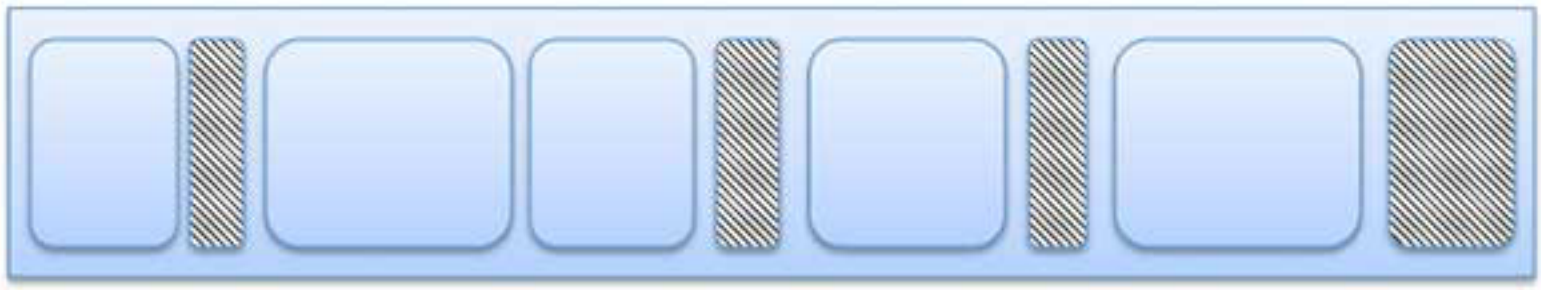
It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused. This problem is known as Fragmentation.

Fragmentation is of two types −

S.N.	Fragmentation & Description
1	**External fragmentation** Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.
2	**Internal fragmentation** Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process.

The following diagram shows how fragmentation can cause waste of memory and a compaction technique can be used to create more free memory out of fragmented memory −

Fragmented memory before compaction



Memory after compaction

External fragmentation can be reduced by compaction or shuffle memory contents to place all free memory together in one large block. To make compaction feasible, relocation should be dynamic.

The internal fragmentation can be reduced by effectively assigning the smallest partition but large enough for the process.

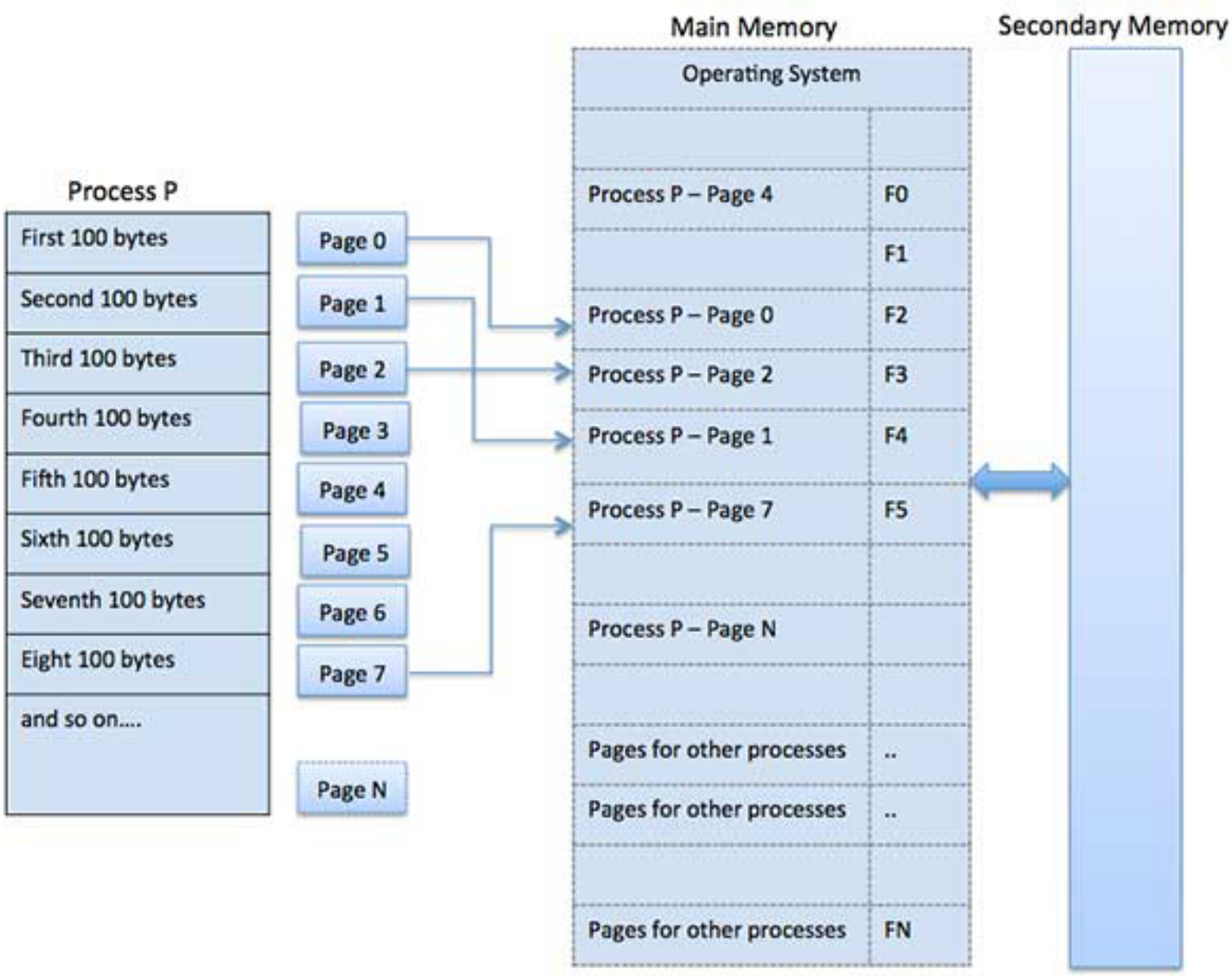
PAGING

PAGING INTRODUCTION

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard that's set up to emulate the computer's RAM. Paging technique plays an important role in implementing virtual memory.

Paging is a memory management technique in which process address space is broken into blocks of the same size called **pages** (size is power of 2, between 512 bytes and 8192 bytes). The size of the process is measured in the number of pages.

imilarly, main memory is divided into small fixed-sized blocks of (physical) memory called **frames** and the size of a frame is kept the same as that of a page to have optimum utilization of the main memory and to avoid external fragmentation.



Address Translation

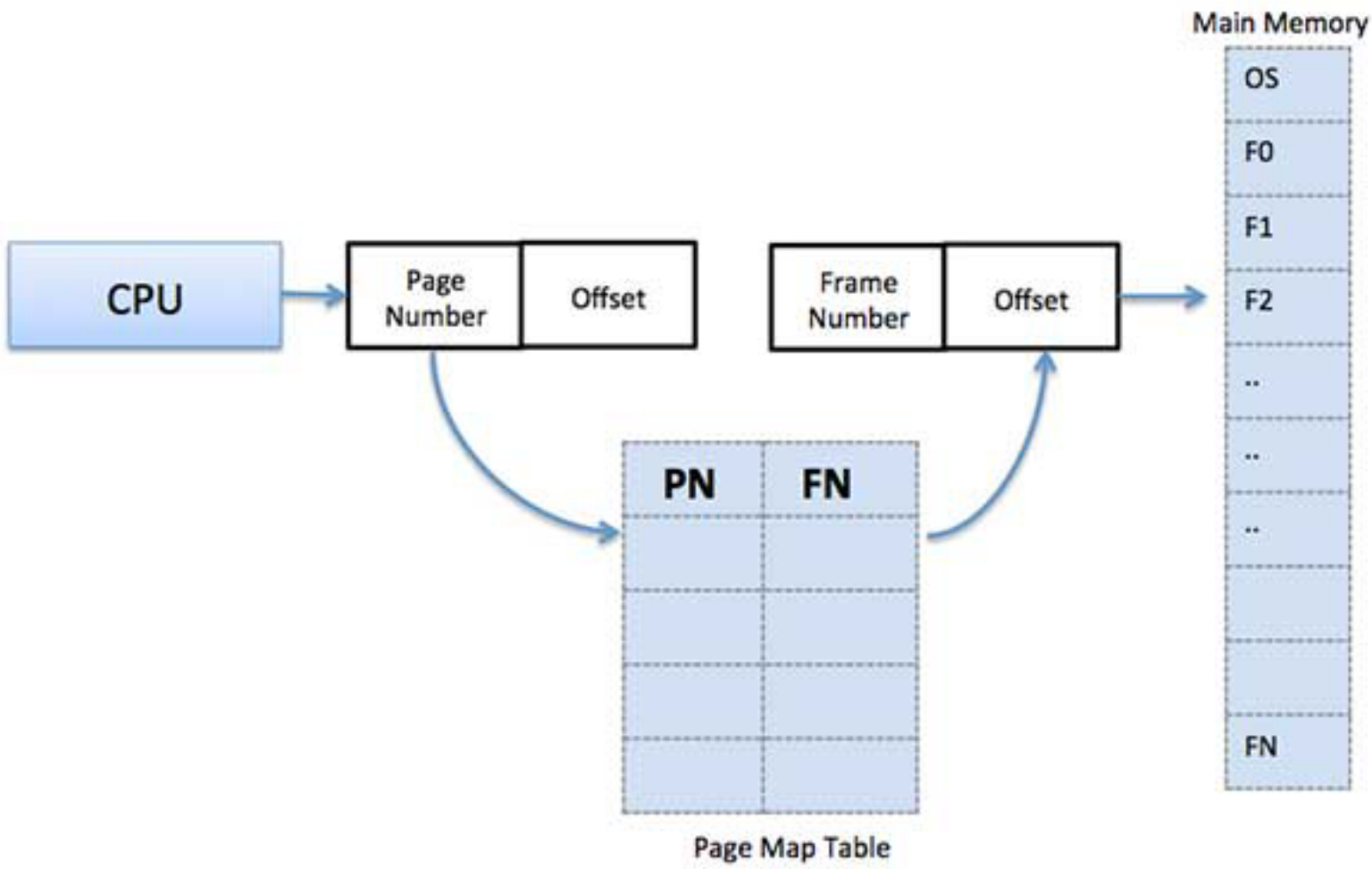
Page address is called **logical address** and represented by **page number**and the **offset**.

Logical Address = Page number + page offset

Frame address is called **physical address** and represented by a **frame number** and the **offset**.

Physical Address = Frame number + page offset

A data structure called **page map table** is used to keep track of the relation between a page of a process to a frame in physical memory.



When the system allocates a frame to any page, it translates this logical address into a physical address and create entry into the page table to be used throughout execution of the program.

When a process is to be executed, its corresponding pages are loaded into any available memory frames. Suppose you have a program of 8Kb but your memory can accommodate only 5Kb at a given point in time, then the paging concept will come into picture.

When a computer runs out of RAM, the operating system (OS) will move idle or unwanted pages of memory to secondary memory to free up RAM for other processes and brings them back when needed by the program.

This process continues during the whole execution of the program where the OS keeps removing idle pages from the main memory and write them onto the secondary memory and bring them back when required by the program.

ADVANTAGES AND DISADVANTAGES OF PAGING

Here is a list of advantages and disadvantages of paging −

- Paging reduces external fragmentation, but still suffer from internal fragmentation.
- Paging is simple to implement and assumed as an efficient memory

management technique.

- Due to equal size of the pages and frames, swapping becomes very easy.
- Page table requires extra memory space, so may not be good for a system having small RAM.

SEGMENTATION

SEGMENTATION INTRODUCTION

Segmentation is a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions. Each segment is actually a different logical address space of the program.

When a process is to be executed, its corresponding segmentation are loaded into non-contiguous memory though every segment is loaded into a contiguous block of available memory.

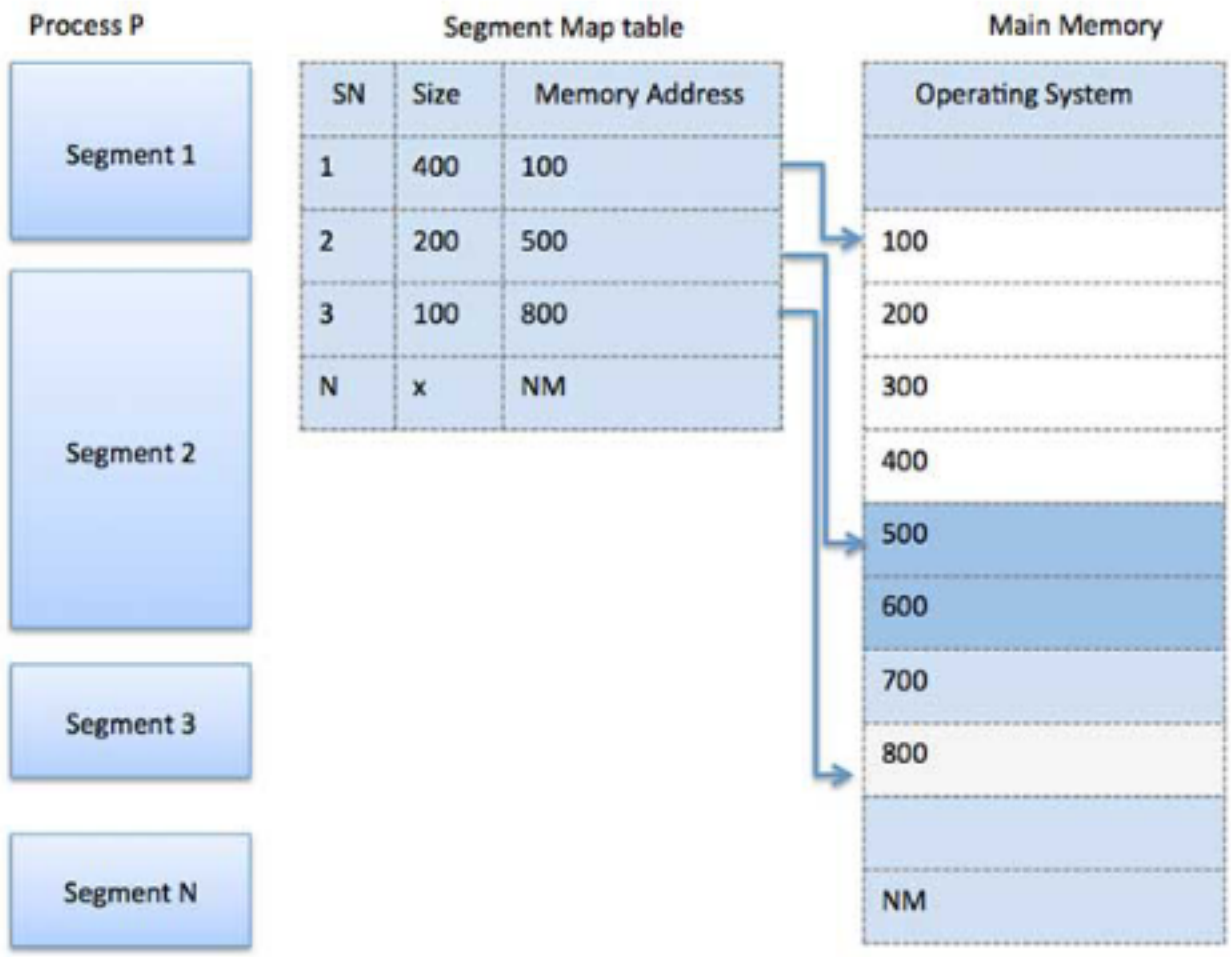
Segmentation memory management works very similar to paging but here segments are of variable-length where as in paging pages are of fixed size.

A program segment contains the program's main function, utility functions, data structures, and so on.

The operating system maintains a **segment map table** for every process and a list of free memory blocks along with segment numbers, their size and corresponding memory locations in main memory.

For each segment, the table stores the starting address of the segment and the length of the segment. A reference to a memory location includes a value that identifies a segment and an offset.

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called **virtual memory** and it is a section of a hard disk that's set up to emulate the computer's RAM.



The main visible advantage of this scheme is that programs can be larger than physical memory. Virtual memory serves two purposes.

First, it allows us to extend the use of physical memory by using disk. Second, it allows us to have memory protection, because each virtual address is translated to a physical address.

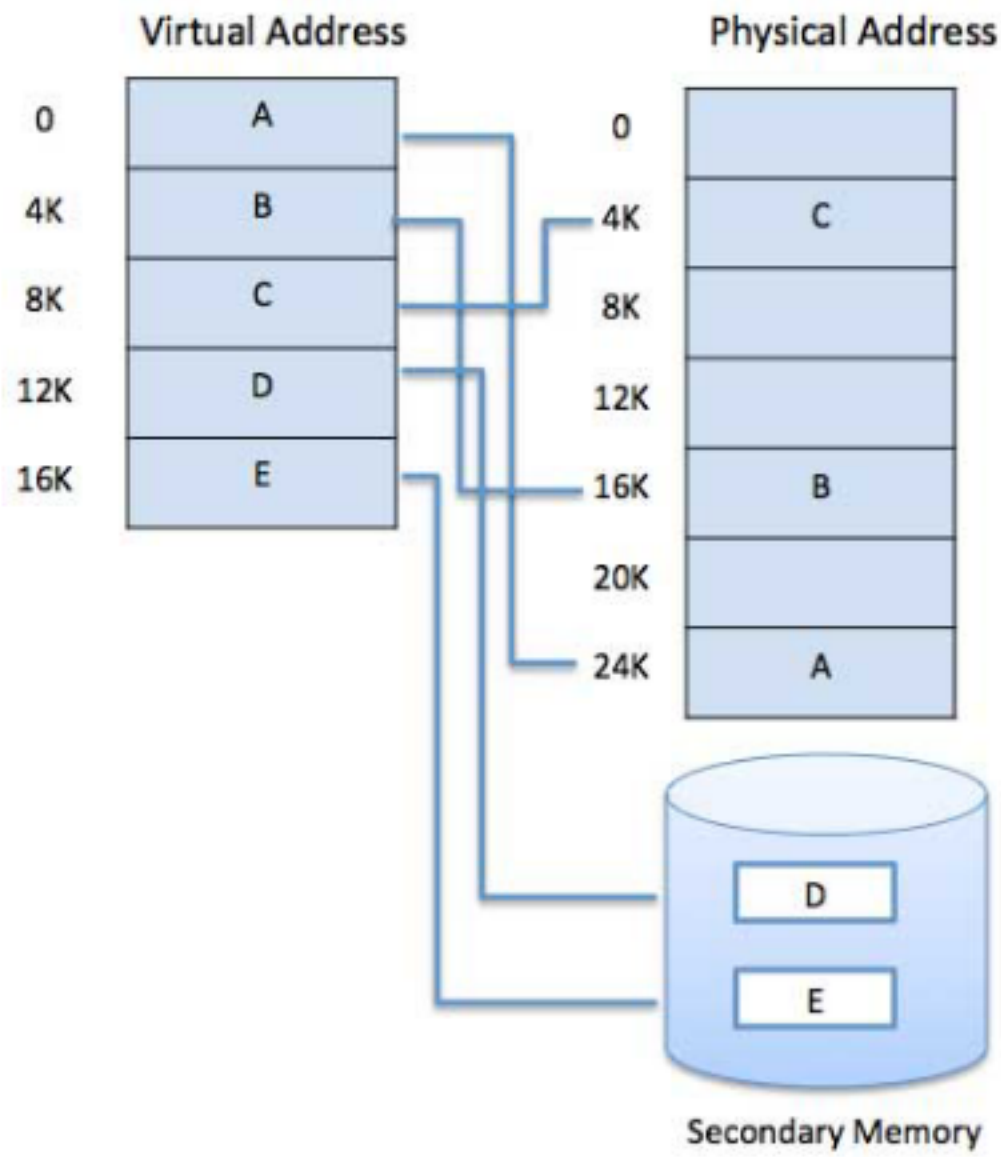
Following are the situations, when entire program is not required to be loaded fully in main memory.

- User written error handling routines are used only when an error occurred in the data or computation.
- Certain options and features of a program may be used rarely.
- 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.
- Less number of I/O would be needed to load or swap each user program

into memory.

- A program would no longer be constrained by the amount of physical memory that is available.
- 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.

Modern microprocessors intended for general-purpose use, a memory management unit, or MMU, is built into the hardware. The MMU's job is to translate virtual addresses into physical addresses. A basic example is given below −



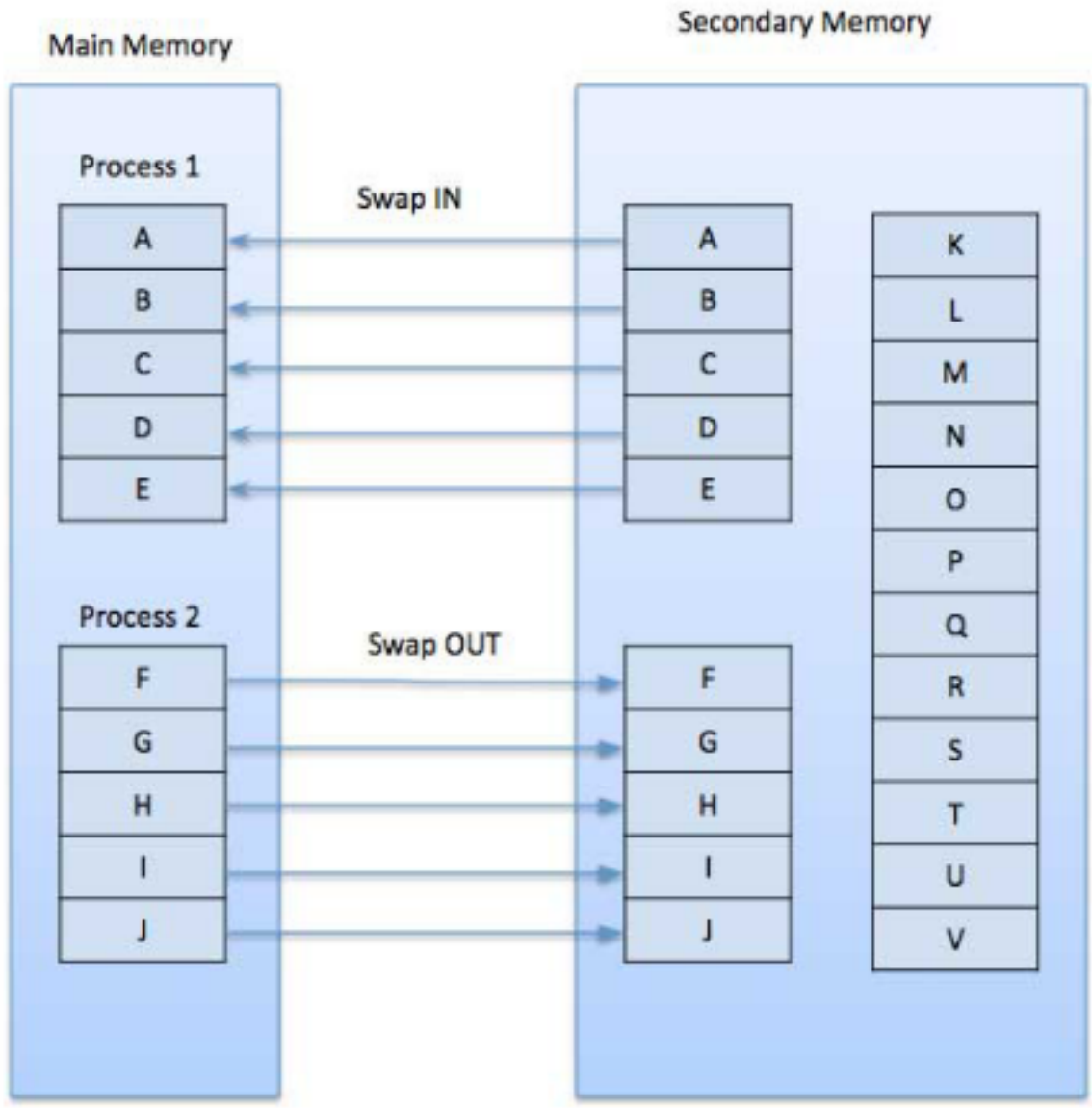
Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory.

DEMAND PAGING

A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand,

not in advance.

When a context switch occurs, the operating system does not copy any of the old program's pages out to the disk or any of the new program's pages into the main memory Instead, it just begins executing the new program after loading the first page and fetches that program's pages as they are referenced.



While executing a program, if the program references a page which is not available in the main memory because it was swapped out a little ago, the processor treats this invalid memory reference as a **page fault** and transfers control from the program to the operating system to demand the page back into the memory.

ADVANTAGES

Following are the advantages of Demand Paging –

- Large virtual memory.
- More efficient use of memory.
- There is no limit on degree of multiprogramming.

DISADVANTAGES

- Number of tables and the amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.

PAGE REPLACEMENT ALGORITHM

Page replacement algorithms are the techniques using which an Operating System decides which memory pages to swap out, write to disk when a page of memory needs to be allocated.

Paging happens whenever a page fault occurs and a free page cannot be used for allocation purpose accounting to reason that pages are not available or the number of free pages is lower than required pages.

When the page that was selected for replacement and was paged out, is referenced again, it has to read in from disk, and this requires for I/O completion.

This process determines the quality of the page replacement algorithm: the lesser the time waiting for page-ins, the better is the algorithm.

A page replacement algorithm looks at the limited information about accessing the pages provided by hardware, and tries to select which pages should be replaced to minimize the total number of page misses, while balancing it with the costs of primary storage and processor time of the algorithm itself.

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,

REFERENCE STRING

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, where we note two things.

- For a given page size, we need to consider only the page number, not the entire address.
- If we have a reference to a page **p**, then any immediately following references to page **p** will never cause a page fault. Page p will be in memory after the first reference; the immediately following references will not fault.

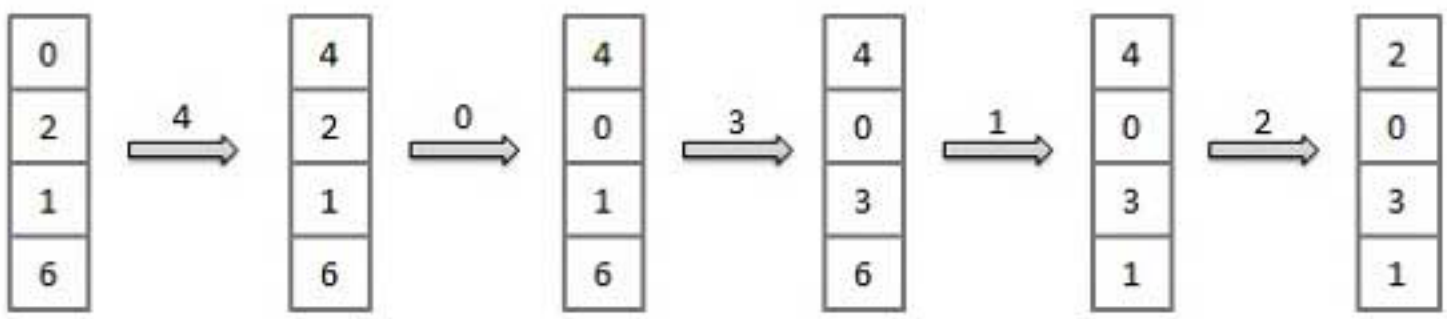
- For example, consider the following sequence of addresses – 123,215,600,1234,76,96
- If page size is 100, then the reference string is 1,2,6,12,0,0

FIRST IN FIRST OUT (FIFO) ALGORITHM

- Oldest page in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages from the tail and add new pages at the head.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x x



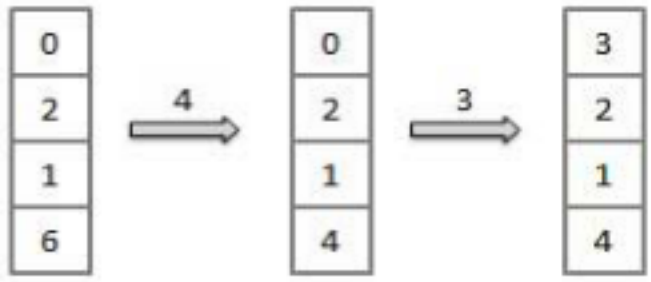
Fault Rate = 9 / 12 = 0.75

OPTIMAL PAGE 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.
- Replace the page that will not be used for the longest period of time. Use the time when a page is to be used.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x



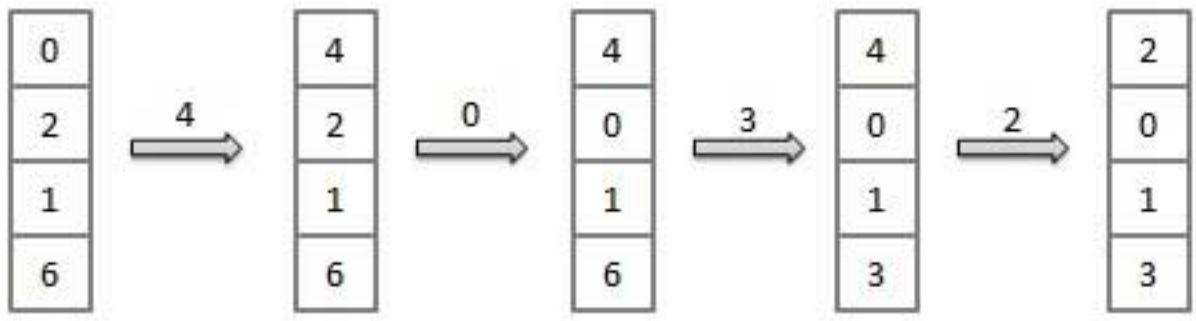
Fault Rate = 6 / 12 = 0.50

LEAST RECENTLY USED (LRU) ALGORITHM

- Page which has not been used for the longest time in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages by looking back into time.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x



Fault Rate = 8 / 12 = 0.67

PAGE BUFFERING ALGORITHM

- To get a process start quickly, keep a pool of free frames.
- On page fault, select a page to be replaced.
- Write the new page in the frame of free pool, mark the page table and restart the process.

- Now write the dirty page out of disk and place the frame holding replaced page in free pool.

LEAST FREQUENTLY USED(LFU) ALGORITHM

- The page with the smallest count is the one which will be selected for replacement.
- 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.

MOST FREQUENTLY USED(MFU) ALGORITHM

- This algorithm is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

UNIT-V

FILE INTRODUCTION

A file is a named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tapes and optical disks. In general, a file is a sequence of bits, bytes, lines or records whose meaning is defined by the files creator and user.

FILE STRUCTURE

A File Structure should be according to a required format that the operating system can understand.

- A file has a certain defined structure according to its type.
- A text file is a sequence of characters organized into lines.
- A source file is a sequence of procedures and functions.
- An object file is a sequence of bytes organized into blocks that are understandable by the machine.
- When operating system defines different file structures, it also contains the code to support these file structure. Unix, MS-DOS support minimum number of file structure.

FILE TYPE

File type refers to the ability of the operating system to distinguish different types of file

such as text files source files and binary files etc. Many operating systems support many types of files. Operating system like MS-DOS and UNIX have the following types of files −

ORDINARY FILES

- These are the files that contain user information.
- These may have text, databases or executable program.
- The user can apply various operations on such files like add, modify, delete or even remove the entire file.

DIRECTORY FILES

- These files contain list of file names and other information related to these files.

Special files

- These files are also known as device files.
- These files represent physical device like disks, terminals, printers, networks, tape drive etc.

These files are of two types −

- **Character special files** − data is handled character by character as in case of terminals or printers.
- **Block special files** − data is handled in blocks as in the case of disks and tapes.

FILE ACCESS MECHANISMS

File access mechanism refers to the manner in which the records of a file may be accessed. There are several ways to access files −

- Sequential access
- Direct/Random access
- Indexed sequential access

SEQUENTIAL ACCESS

A sequential access is that in which the records are accessed in some sequence, i.e., the information in the file is processed in order, one record after the other. This access method is the most primitive one. Example: Compilers usually access files in this fashion.

DIRECT/RANDOM ACCESS

- Random access file organization provides, accessing the records directly.
- Each record has its own address on the file with by the help of which it can be directly accessed for reading or writing.
- The records need not be in any sequence within the file and they need not be in adjacent locations on the storage medium.

INDEXED SEQUENTIAL ACCESS

- This mechanism is built up on base of sequential access.
- An index is created for each file which contains pointers to various blocks.
- Index is searched sequentially and its pointer is used to access the file directly.

SPACE ALLOCATION

Files are allocated disk spaces by operating system. Operating systems deploy following three main ways to allocate disk space to files.

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

CONTIGUOUS ALLOCATION

- Each file occupies a contiguous address space on disk.
- Assigned disk address is in linear order.
- Easy to implement.
- External fragmentation is a major issue with this type of allocation technique.

LINKED ALLOCATION

- Each file carries a list of links to disk blocks.
- Directory contains link / pointer to first block of a file.
- No external fragmentation
- Effectively used in sequential access file.
- Inefficient in case of direct access file.

INDEXED ALLOCATION

- Provides solutions to problems of contiguous and linked allocation.
- A index block is created having all pointers to files.
- Each file has its own index block which stores the addresses of disk space

occupied by the file.

- Directory contains the addresses of index blocks of files.

FILE DIRECTORIES:

Collection of files is a file directory. The directory contains information about the files, including attributes, location and ownership. Much of this information, especially that is concerned with storage, is managed by the operating system. The directory is itself a file, accessible by various file management routines.

Information contained in a device directory are:

- Name
- Type
- Address
- Current length
- Maximum length
- Date last accessed
- Date last updated
- Owner id
- Protection information

Operation performed on directory are:

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

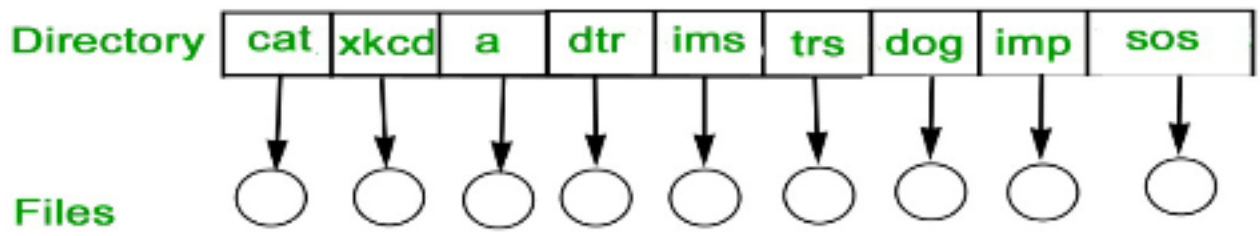
Advantages of maintaining directories are:

- **Efficiency:** A file can be located more quickly.
- **Naming:** It becomes convenient for users as two users can have same name for different files or may have different name for same file.
- **Grouping:** Logical grouping of files can be done by properties e.g. all java programs, all games etc.

SINGLE-LEVEL DIRECTORY

In this a single directory is maintained for all the users.

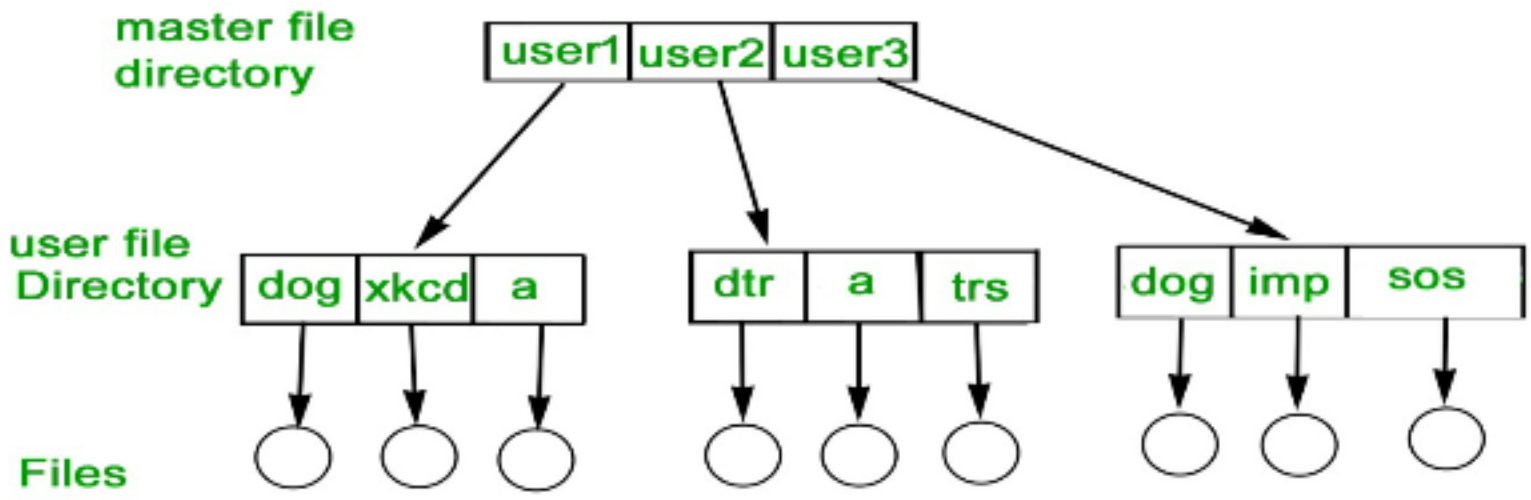
- **Naming problem:** Users cannot have same name for two files.
- **Grouping problem:** Users cannot group files according to their need.



TWO-LEVEL DIRECTORY

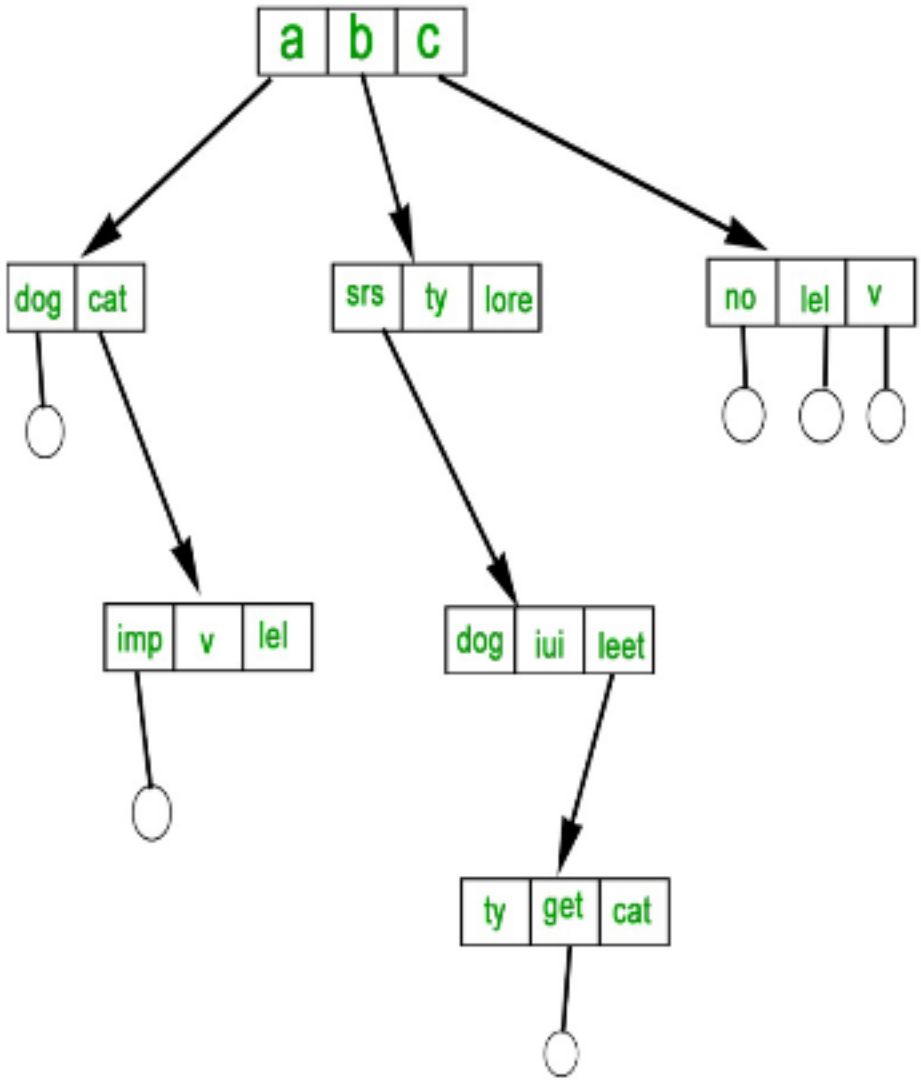
In this separate directories for each user is maintained.

- Path name:Due to two levels there is a path name for every file to locate that file.
- Now,we can have same file name for different user.
- Searching is efficient in this method.



TREE-STRUCTURED DIRECTORY :

Directory is maintained in the form of a tree. Searching is efficient and also there is grouping
capability. We have absolute or relative path name for a file.



PROTECTION

- ★ When information is kept in a computer system, a major concern is its protection from both physical damage (reliability) and improper access (protection).
- ★ File systems can be damaged by hardware problems (such as errors reading and writing) , power surges or failures, hard crashes, dirt, temperature extremes and vandalism.
- ★ Files may be deleted accidentally. Bugs in the file system software can also cause file contents to be lost.
- ★ Protection can be provided in many ways. For a small single user system, we might provide protection by physically removing the floppy disks.
- ★ In a multi user system , however other mechanisms are needed.

TYPES OF ACCESS

- ★ The need for protecting files is a direct result of the ability to access files.
- ★ Protection mechanisms provide controlled access by limiting the types of file

access that can be made

★ Access is permitted or denied depending on several factors, one of which is the type of access requested.

★ A file has many properties so to defile the file property operating system provides a lot of information that can be performed on the file.

★ There are many simple and easy operations of file like create, delete, update and some others like rename.

1. **Creating a file** - for creating any file there are mainly two steps first the free space is available in the system and second is the new entry of file must be made in the directory.
2. **Writing a file** - For writing a file the system call specifies the two things name of the file and the information that is written on the file. When the name of the file is given the system search that file in the directory, the write pointer point the location where the next write is to take place and write pointer is updated.
3. **Reading a file** - For reading a file the system call also specifies the two things name of the file and the read pointer. The directory is searched for the given entry and system keep track on the read pointer. The read pointer is updated once the read is completed. The same pointer is used by both the read and write operation on the file.
4. **Deleting a file** - For deleting a file first we search for the directory and then erase the directory. After that, we release the space so that it can be reused by another file.
5. **Repositioning of the file** - First, the directory is searched for the file and the current position of the file is changed by the new position.

FILE ALLOCATION METHODS

The allocation methods define how the files are stored in the disk blocks. There are three main disk space or file allocation methods.

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

The main idea behind these methods is to provide:

- Efficient disk space utilization.

- Fast access to the file blocks.

All the three methods have their own advantages and disadvantages as discussed below:

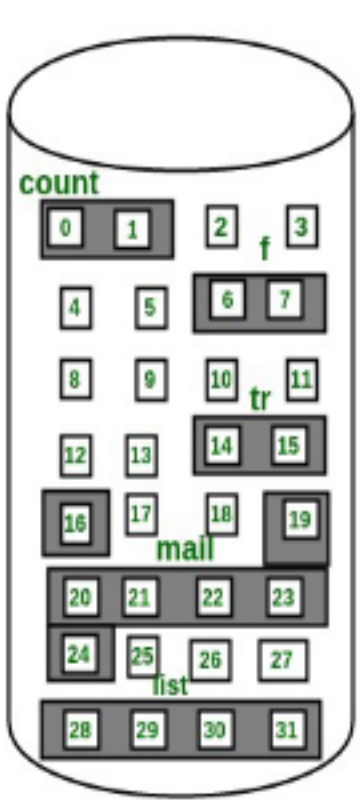
1. CONTIGUOUS ALLOCATION

In this scheme, each file occupies a contiguous set of blocks on the disk. For example, if a file requires n blocks and is given a block b as the starting location, then the blocks assigned to the file will be: b, b+1, b+2,……b+n-1. This means that given the starting block address and the length of the file (in terms of blocks required), we can determine the blocks occupied by the file.

The directory entry for a file with contiguous allocation contains

- Address of starting block
- Length of the allocated portion.

The file 'mail' in the following figure starts from the block 19 with length = 6 blocks. Therefore, it occupies 19, 20, 21, 22, 23, 24 blocks.



Directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Advantages:

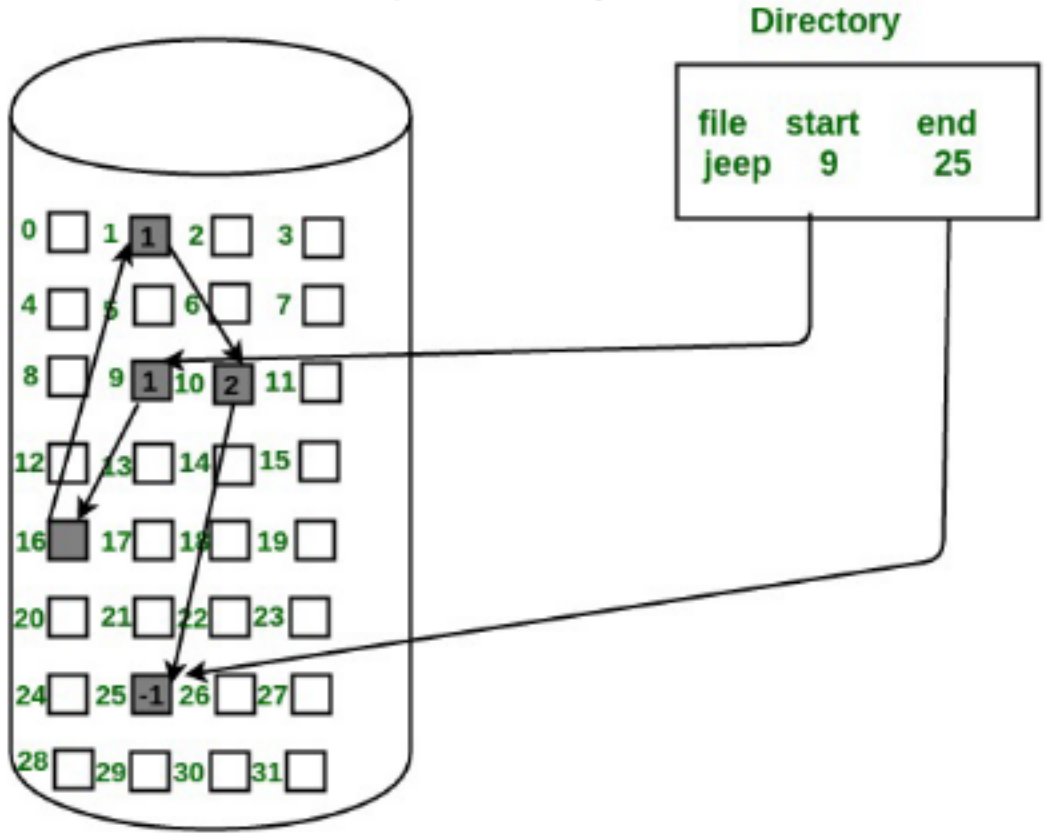
- Both the Sequential and Direct Accesses are supported by this. For direct access, the address of the kth block of the file which starts at block b can easily be obtained as (b+k).
- This is extremely fast since the number of seeks are minimal because of contiguous allocation of file blocks.

Disadvantages:

- This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.
- Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance

2. LINKED LIST ALLOCATION

- In this scheme, each file is a linked list of disk blocks which need not be contiguous. The disk blocks can be scattered anywhere on the disk. The directory entry contains a pointer to the starting and the ending file block. Each block contains a pointer to the next block occupied by the file.
- The file 'jeep' in following image shows how the blocks are randomly distributed. The last block (25) contains -1 indicating a null pointer and does not point to any other block.



Advantages:

- This is very flexible in terms of file size. File size can be increased easily since the system does not have to look for a contiguous chunk of memory.
- This method does not suffer from external fragmentation. This makes it relatively better in terms of memory utilization.

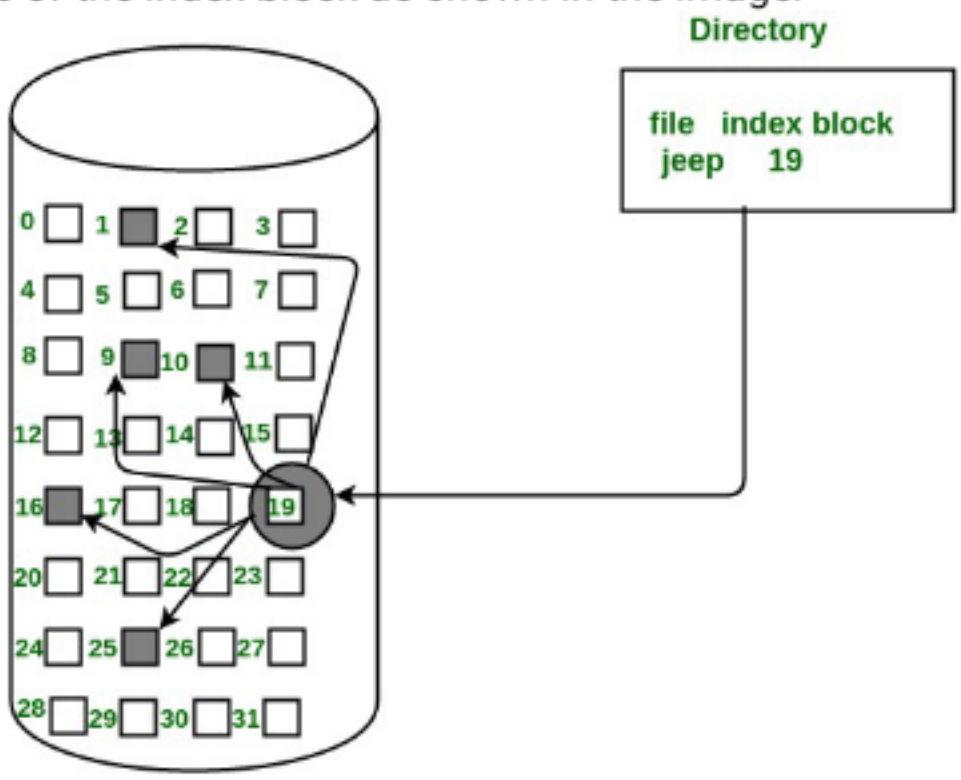
Disadvantages:

- Because the file blocks are distributed randomly on the disk, a large number of seeks are needed to access every block individually. This makes linked allocation slower.
- It does not support random or direct access. We can not directly access the blocks of a file. A block k of a file can be accessed by traversing k blocks sequentially (sequential access) from the starting block of the file via block pointers.
- Pointers required in the linked allocation incur some extra overhead

3. INDEXED ALLOCATION

In this scheme, a special block known as the Index block contains the pointers to all the blocks occupied by a file. Each file has its own index block. The ith entry in the index block contains the disk address of the ith file block. The directory entry contains

the address of the index block as shown in the image:



Advantages:

- This supports direct access to the blocks occupied by the file and therefore provides fast access to the file blocks.
- It overcomes the problem of external fragmentation.

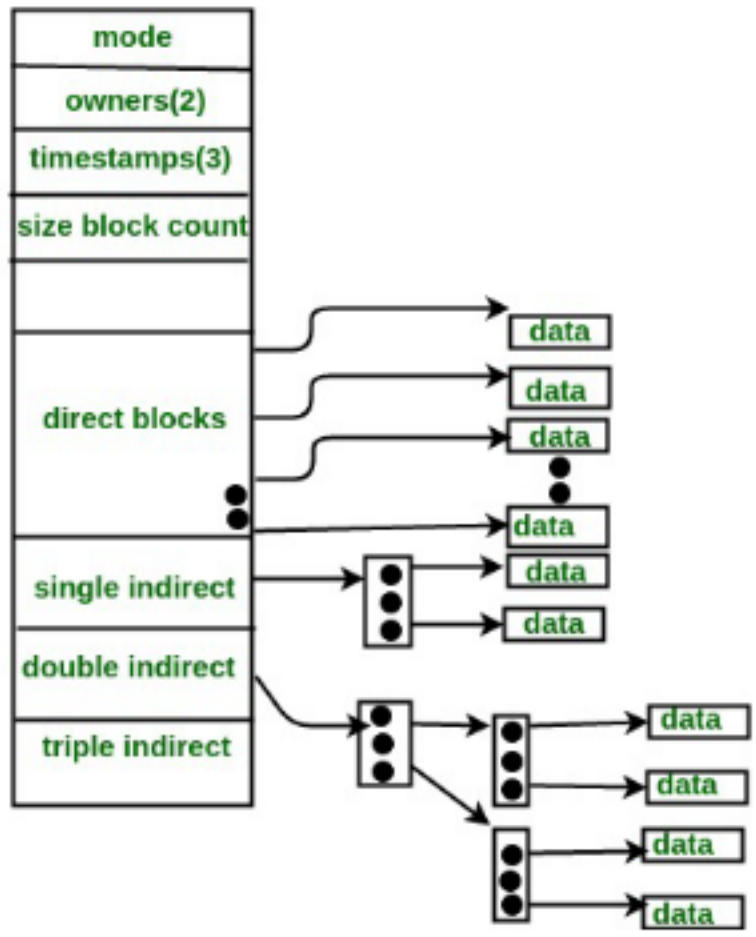
Disadvantages:

- The pointer overhead for indexed allocation is greater than linked allocation.
- For very small files, say files that expand only 2-3 blocks, the indexed allocation would keep one entire block (index block) for the pointers which is inefficient in terms of memory utilization. However, in linked allocation we lose the space of only 1 pointer per block

For files that are very large, single index block may not be able to hold all the pointers. Following mechanisms can be used to resolve this:

1. **Linked scheme:** This scheme links two or more index blocks together for holding the pointers. Every index block would then contain a pointer or the address to the next index block.
2. **Multilevel index:** In this policy, a first level index block is used to point to the second level index blocks which inturn points to the disk blocks occupied by the file. This can be extended to 3 or more levels depending on the maximum file size.
3. **Combined Scheme:** In this scheme, a special block called the **Inode (information Node)** contains all the information about the file such as the name, size, authority, etc and the remaining space of Inode is used to store the Disk Block addresses which contain the actual file *as shown in the image below.* The first few of these pointers in Inode point to the **direct blocks** i.e the pointers contain the addresses of the disk blocks that contain data of the file. The next few pointers point to indirect blocks. Indirect blocks may be single indirect, double indirect or triple indirect. **Single Indirect block** is the disk block that does not contain the file data but the disk address of the blocks that contain the file data.

Similarly, **double indirect blocks** do not contain the file data but the disk address of the blocks that contain the address of the blocks containing the file data.



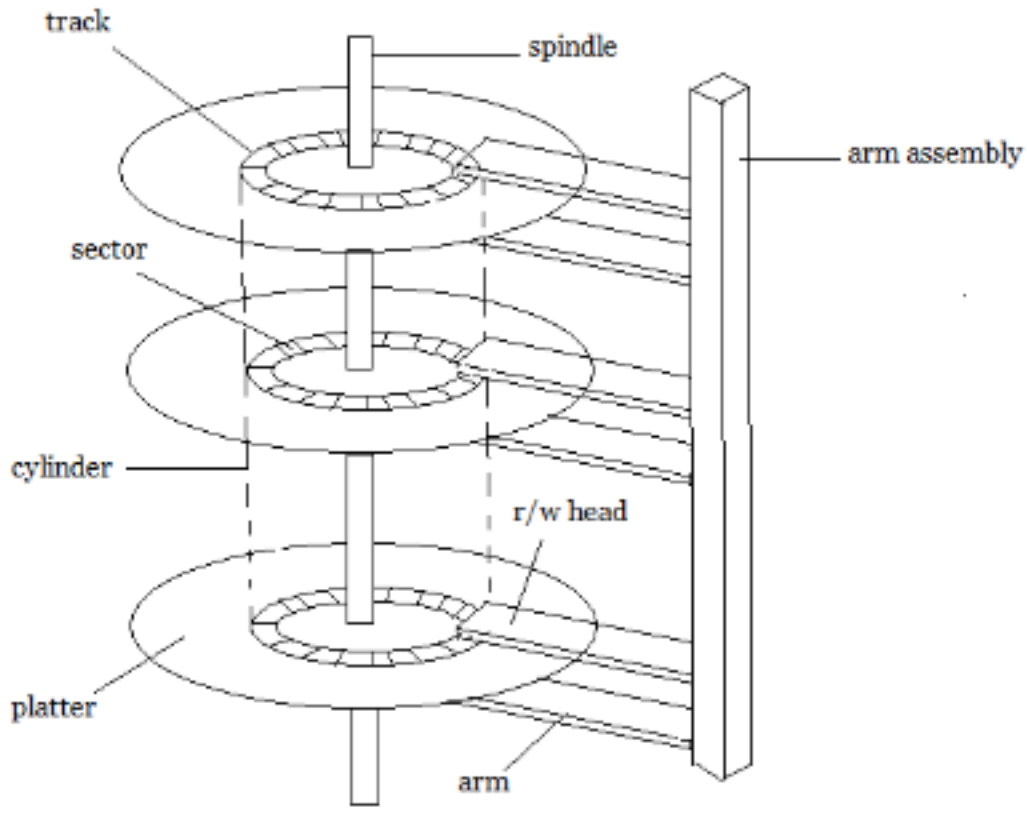
SECONDARY STORAGE AND DISK SCHEDULING ALGORITHMS

Secondary storage devices are those devices whose memory is non volatile, meaning, the stored data will be intact even if the system is turned off. Here are a few things worth noting about secondary storage.

- Secondary storage is also called auxiliary storage.
- Secondary storage is less expensive when compared to primary memory like RAMs.
- The speed of the secondary storage is also lesser than that of primary storage.
- Hence, the data which is less frequently accessed is kept in the secondary storage.
- A few examples are magnetic disks, magnetic tapes, removable thumb drives etc.

MAGNETIC DISK STRUCTURE

In modern computers, most of the secondary storage is in the form of magnetic disks. Hence, knowing the structure of a magnetic disk is necessary to understand how the data in the disk is accessed by the computer.



STRUCTURE OF A MAGNETIC DISK

A magnetic disk contains several **platters**. Each platter is divided into circular shaped **tracks**.

The length of the tracks near the centre is less than the length of the tracks farther from the centre.

Each track is further divided into **sectors**, as shown in the figure.
Tracks of the same distance from centre form a cylinder.

A read-write head is used to read data from a sector of the magnetic disk.

The speed of the disk is measured as two parts:

- **Transfer rate:** This is the rate at which the data moves from disk to the computer.
- **Random access time:** It is the sum of the seek time and rotational latency.

Seek time is the time taken by the arm to move to the required track. **Rotational latency** is defined as the time taken by the arm to reach the required sector in the track.

Even though the disk is arranged as sectors and tracks physically, the data is logically arranged and addressed as an array of blocks of fixed size. The size of a block can be **512** or **1024** bytes. Each logical block is mapped with a sector on the disk, sequentially. In this way, each sector in the disk will have a logical address.

DISK SCHEDULING ALGORITHMS

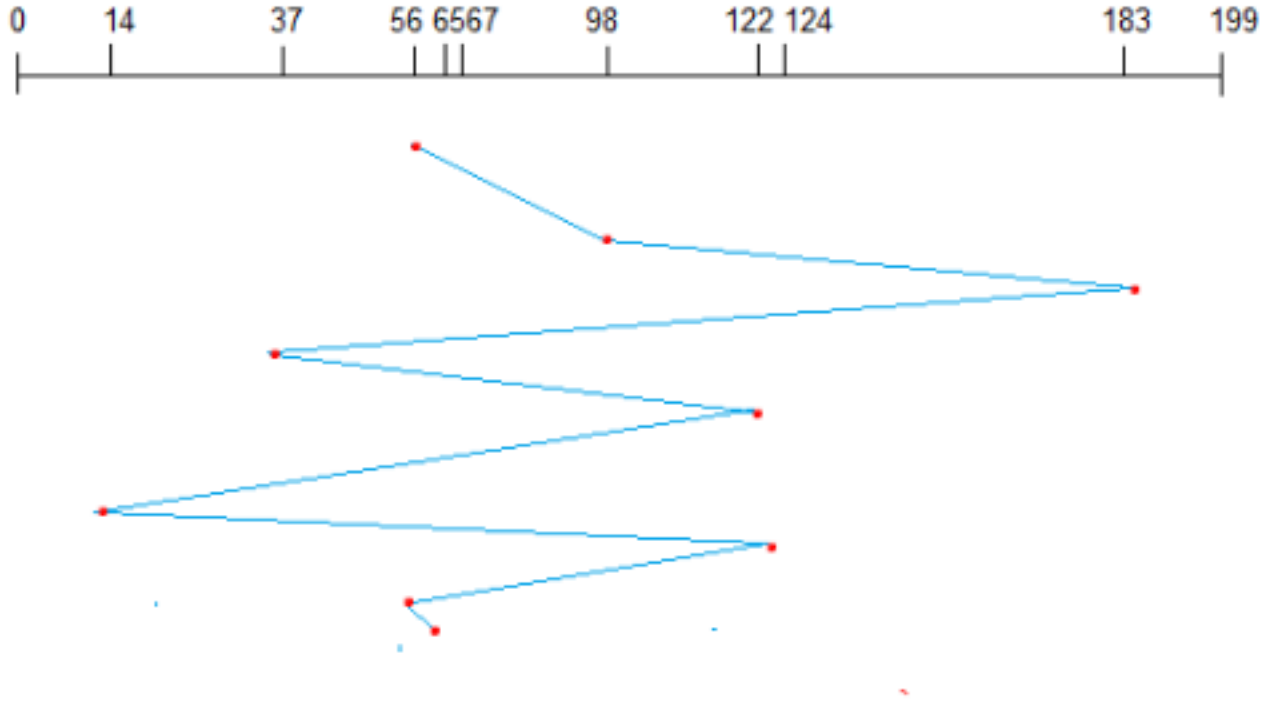
On a typical multiprogramming system, there will usually be multiple disk access requests at any point of time. So those requests must be scheduled to achieve good efficiency. Disk scheduling is similar to process scheduling. Some of the disk scheduling algorithms are described below.

FIRST COME FIRST SERVE

This algorithm performs requests in the same order asked by the system. Let's take an example where the queue has the following requests with cylinder numbers as follows:

98, 183, 37, 122, 14, 124, 65, 67

Assume the head is initially at cylinder **56**. The head moves in the given order in the queue i.e., **56 → 98 → 183 → ... → 67**.



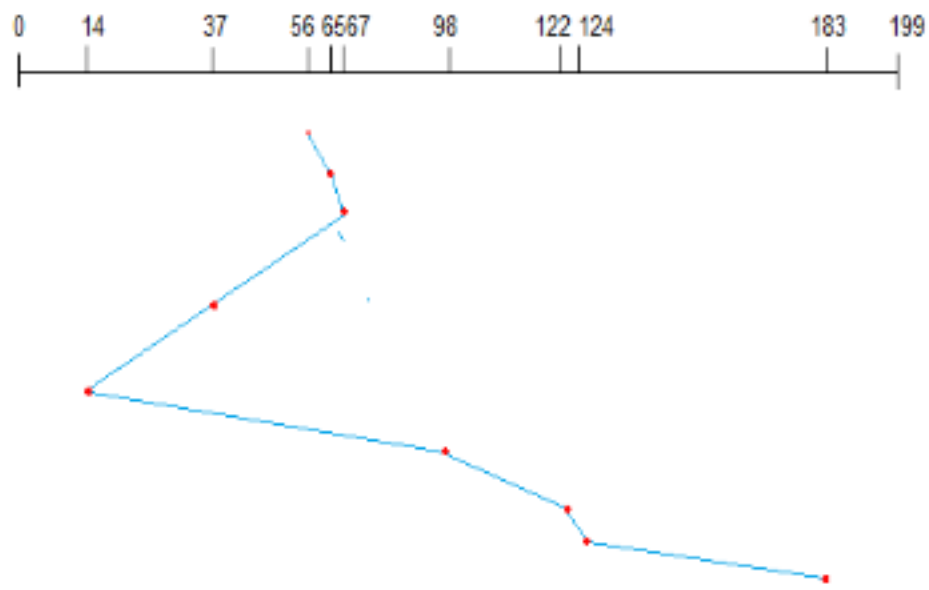
SHORTEST SEEK TIME FIRST (SSTF)

Here the position which is closest to the current head position is chosen first. Consider the previous example where disk queue looks like,

98, 183, 37, 122, 14, 124, 65, 67

Assume the head is initially at cylinder **56**. The next closest cylinder to **56** is **65**,

and then the next nearest one is **67**, then **37**, **14**, so on.



SCAN ALGORITHM

This algorithm is also called the elevator algorithm because of it's behavior. Here, first the head moves in a direction (say backward) and covers all the requests in the path.

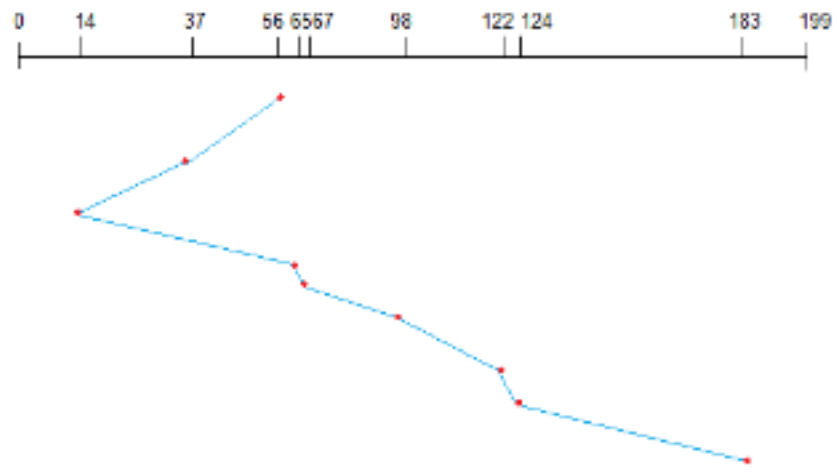
Then it moves in the opposite direction and covers the remaining requests in the path. This behavior is similar to that of an elevator. Let's take the previous example,

98, 183, 37, 122, 14, 124, 65, 67

Assume the head is initially at cylinder **56**. The head moves in backward direction and accesses **37** and **14**. Then it goes in the opposite direction and accesses the cylinders as they come in the path.

C-SCAN Disk Scheduling Algorithm-

- ★ Circular-SCAN Algorithm is an improved version of the SCAN Algorithm.
- ★ Head starts from one end of the disk and move towards the other end servicing all the requests in between.
- ★ After reaching the other end, head reverses its direction.
- ★ It then returns to the starting end without servicing any request in between.
- ★ The same process repeats.



LOOK SCHEDULING

The main difference between SCAN Algorithm and LOOK Algorithm is-

- SCAN Algorithm scans all the cylinders of the disk starting from one end to the other end even if there are no requests at the ends.
- LOOK Algorithm scans all the cylinders of the disk starting from the first request at one end to the last request at the other end.

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".
BOOT BLOCK initializes system
. – The bootstrap is stored in ROM.
– Bootstrap loader program brings in full bootstrap program from disk.
• Bootstrap program stored at fixed location on disk (boot blocks)
• Allows updating bootstrap program.

DISK MANAGEMENT: BAD BLOCKS

• Methods such as sector sparing (also known as forwarding) used to handle bad blocks.
– Spare sectors set aside on low-level formatting
– Controller told to replace a bad sector logically with one of the spare sectors
– To retain effectiveness of disk-scheduling optimization, provide spare sectors in each cylinder and also provide some spare cylinders. Use spare sector from same cylinder if possible

SECTOR SLIPPING

– moves blocks following bad block downward (occupying spare sector) to free up block following bad block

– skips bad block, using freed up block to hold that sector's information.