Code: 23CS3401 **PVP 23**

II BTech – II Semester – Regular / Supplementary Examinations
APRIL 2026
OPERATING SYSTEMS
(COMPUTER SCIENCE & ENGINEERING)
Max. Marks: 70

SCHEME OF EVALUATION

PART-A

1.a) Define Operating System from the user view and system view. 2M

An Operating System (OS) is system software that acts as an interface between the user and the computer hardware. It can be understood from two viewpoints:

The user view depends on the type of system being used and focuses mainly on convenience, ease of use, and performance.

From the system viewpoint, the OS acts as:

1. Resource Allocator
2. Control Program

1.b) List any four operating system services. 2M

- Program Execution
- User Interface
- Error Detection
- I/O Operations
- File-System Manipulation
- Communication
- Resource Allocation
- Security and Protection
- Accounting

1.c) What is a process? How is it different from a program? 2M

A process is a program that is currently being executed . Process is a active entity.

A program is a set of instructions stored on disk. Program is a passive entity. A process is the actual execution of those instructions with system resources allocated to it.

1.d) what is a process control block mention any two fields stored in it? 2M

A Process Control Block (PCB) is a data structure maintained by the Operating System to store all information related to a process.Whenever a process is created, the OS creates a PCB for it. Fields in PCB:

- Process State
- Process ID (PID)
- Program Counter
- CPU Registers
- CPU Scheduling Information
- Memory Management Information
- Accounting Information
- I/O Status Information

1.e) what is a critical section problem? State its requirements. 2M

Consider a system consisting of *n* processes $\{P_0, P_1, ..., P_{n-1}\}$. Each process has a segment of code, called critical section, in which the process may be changing common variables, updating a table, writing a file, and so on.

A critical section is a part of a program where shared resources such as Variables, Files , Memory , Data structures are accessed or modified.

Requirements of Critical Section Problem

1. Mutual Exclusion

2. Progress

3. Bounded Waiting

1.f) Define Mutex Lock. How is it different from a semaphore? 2M

A Mutex Lock (Mutual Exclusion Lock) is a synchronization mechanism used to protect shared resources from being accessed simultaneously by multiple processes or threads. Mutex lock allows only one process/thread to access a resource at a time with ownership, whereas a semaphore is a signaling mechanism that can allow multiple processes to access resources based on its count.

1.g) What is address binding? Name its different types. 2M

Address binding is the process of mapping a program's logical addresses to physical memory addresses.

In other words, it is the method of converting the addresses generated by a program into actual locations in main memory.

Types of Address Binding

1. Compile-Time Binding
2. Load-Time Binding
3. Execution-Time (Run-Time) Binding

1.h) Define Paging.What is a page frame? 2M

Paging is a memory management technique in which:

- Physical memory is divided into fixed-size blocks called frames
- Logical memory is divided into fixed-size blocks called pages

A page frame (or simply frame) is a fixed-size block of physical memory used to store pages of a process.

1.i) what is a file? List any two attributes. 2M

A file is a named collection of related information stored on secondary storage devices such as a hard disk.

- Name
- Identifier
- Type
- Location
- Size
- Protection
- Time and Date Information
- Owner/User Identification

1.j) Define Sequential access and direct access methods. 2M

Sequential Access Method

In sequential access, data in a file is accessed one record after another in order.

- Records are processed sequentially from beginning to end.
- To access a particular record, all previous records must be read first.

Direct Access Method

In direct access (or random access), records can be accessed directly without reading previous records.

- Any record can be read or written immediately using its address or key.

PART B

UNIT I

2a) Explain what operating systems do from user view ans system view with examples. 5M

Operating System can be viewed from two viewpoints– User views & System views

User Views:- *T*he user's view of the operating system depends on the type of user.

- If the user is using **stand alone** system, then OS is designed for ease of use and high performances. Here resource utilization is not given importance.
- If the users are at different **terminals** connected to a mainframe or minicomputers, by sharing information and resources, then the OS is designed to maximize resource utilization. OS is designed such that the CPU time, memory and i/o are used efficiently and no single user takes more than the resource allotted to them.
- If the users are in **workstations**, connected to networks and servers, then the user have a system unit of their own and shares resources and files with other systems. Here the OSis designed for both ease of use and resource availability (files).
- Other systems like embedded systems used in home device (like washing m/c) & automobiles do not have any user interaction. There are some LEDs to show the status of its work
- Users of **hand-held** systems, expects the OS to be designed for ease of use and performance per amount of battery life

SystemViews:- Operating system can be viewed as a **resource allocator and control program**.

- **Resource allocator** – The OS acts as a manager of hardware and software resources. CPU time, memory space, file-storage space, I/O devices, shared files etc. are the different resources required during execution of a program. There can be conflicting request for these resources by different programs running in same system. The OS assigns the resources to the requesting program depending on the priority.
- **Control Program** – The OS is a control program and manage the execution of user program to prevent errors and improper use of the computer.

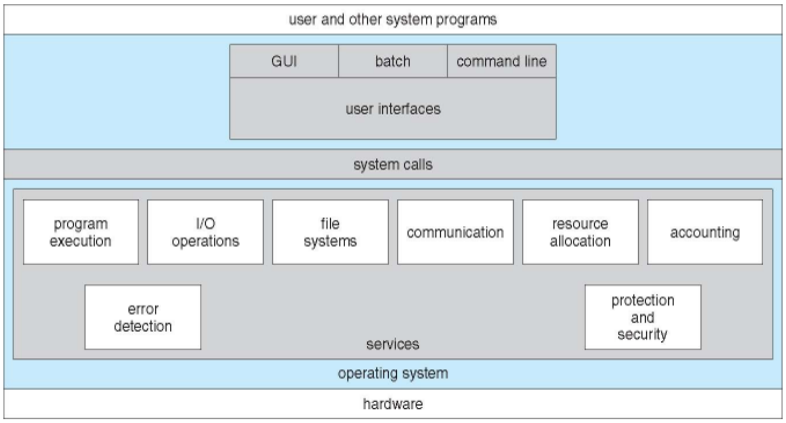
2b) Explain operating system services and illustrate them with neat diagram. 5M

Operating systems provide an environment for execution of programs and services to programs and users. **One set of operating-system services** provides functions that are helpful to the user:

- **User interface** - Almost all operating systems have a user interface (**UI**).
  - Varies between **Command-Line (CLI)**, **Graphics User Interface (GUI)**, **Batch**
- **Program execution** - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
- **I/O operations** - A running program may require I/O, which may involve a file or an I/O device
- **File-system manipulation** - The file system is of particular interest. Programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.
- **Communications** – Processes may exchange information, on the same computer or between computers over a network
  - Communications may be via shared memory or through message passing (packets moved by the OS)
- **Error detection** – OS needs to be constantly aware of possible errors
  - May occur in the CPU and memory hardware, in I/O devices, in user program
  - For each type of error, OS should take the appropriate action to ensure correct and consistent computing
  - Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system

- **Another set of OS functions** exists for ensuring the efficient operation of the system itself via resource sharing
  - **Resource allocation -** When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
    - Many types of resources - CPU cycles, main memory, file storage, I/O devices.

- **Accounting -** To keep track of which users use how much and what kinds of computer resources
- **Protection and security -** The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other
    - **Protection** involves ensuring that all access to system resources is controlled
    - **Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts



A View of Operating System Services

3a) Describe major functions of an operating system including process,memory,storage and protection management. 5M

Process Management

- A program under execution is a process. A process needs resources like CPU time, memory, files, and I/O devices for its execution. These resources are given to the process when it is created or at run time. When the process terminates, the operating system reclaims the resources.
- The program stored on a disk is a **passive entity** and the program under execution is an **active entity**. A single-threaded process has one **program counter** specifying the next instruction to execute. The CPU executes one instruction of the process after another, until the process completes. A multithreaded process has multiple program counters, each pointing to the next instruction to execute for a given thread.

Memory Management

Main memory is a large array of words or bytes. Each word or byte has its own address.

Main memory is the storage device which can be easily and directly accessed by the CPU. As the program executes, the central processor reads instructions and also reads and writes data from main memory.

To improve both the utilization of the CPU and the speed of the computer's response to its users, general-purpose computers must keep several programs in memory, creating a need for memory management.

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

- Keeping track of which parts of memory are currently being used by user.
- Deciding which processes and data to move into and out of memory.
- Allocating and reallocating memory space as needed.

Storage Management

There are three types of storage management

i) File system management

ii) Mass-storage management

iii) Cache management.

File-System Management

- File management is one of the most visible component s of an operating system. Computer scan store information on several different types of physical media. Magnetic disk, optical disk, and magnetic tape are the most common. Each of these media has its own characteristics and physical organization. Each medium is controlled by a device, such as a disk drive or tape drive, that also has its own unique characteristics.
- A file is a collection of related information defined by its creator. Commonly, files represent programs and data. Data files may be numeric, alphabetic, alphanumeric, or binary. Files may be free-form (for example, text files), or they may be formatted rigidly (for example, fixed fields).

Mass-Storage Management

- As the main memory is too small to accommodate all data and programs, and as the data that it holds are erased when power is lost, the computer system must provide secondary storage to back up main memory. Most modern computer systems use disks as the storage medium for both programs and data.
- Most programs—including compilers, assemblers, word processors, editors, and formatters— are stored on a disk until loaded into memory and then use the disk as both the source and destination of their processing. Hence, the proper management of disk storage is of central importance to a computer system.

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

  - Free-space management
  - Storage allocation
  - Disk scheduling

Caching

- **Caching** is an important principle of computer systems. Information is normally kept in some storage system (such as main memory). As it is used, it is copied into a faster storage system— the cache—as temporary data. When a particular piece of information is required, first we check whether it is in the cache. If it is, we use the information directly from the cache; if it is not in cache, we use the information from the source, putting a copy in the cache under the assumption that we will need it again soon.

Protection

- **Protection** is a mechanism for controlling the access of processes or users to the resources defined by a computer system. This mechanism must provide means for specification of the controls to be imposed and means for enforcement.
- Protection improves reliability. A protection-oriented system provides a means to distinguish between authorized and unauthorized usage. A system can have adequate protection but still be prone to failure and allow inappropriate access.

3b) Discuss different computing environments supported by operating systems with examples. 5M

Computing Environments

The different computing environments are-

Traditional Computing

In the latter half of the previous century, computing resources were scarce. Years before, systems were either batch or interactive. Batch system processed jobs in bulk, with predetermined input (from files or other sources of data). Interactive systems waited for input from users. To optimize the use of the computing resources, multiple users shared time on these systems. Time-sharing systems used a timer and scheduling algorithms to rapidly cycle processes through the CPU, giving each user a share of the resources.

- Today, traditional time-sharing systems are used everywhere. The same scheduling technique is still in use on workstations and servers, but frequently the processes are all owned by the same user (or a single user and the operating system). User processes, and system processes that provide services to the user, are managed so that each frequently gets a slice of computer time.

Mobile Computing :

Mobile Computing is possible through Handheld smartphones, tablets, etc.

The Main functional differences between mobile phones and a "traditional" laptop are :

- Mobiles support Extra feature – more OS features (GPS(Global Positioning System) – for finding locations, gyroscope- for orientation,sliding,tilting etc)
- Allows new types of apps like ***augmented reality***
- Use IEEE 802.11 wireless, or cellular data networks for connectivity
- Leaders are **Apple iOS** and **Google Android**

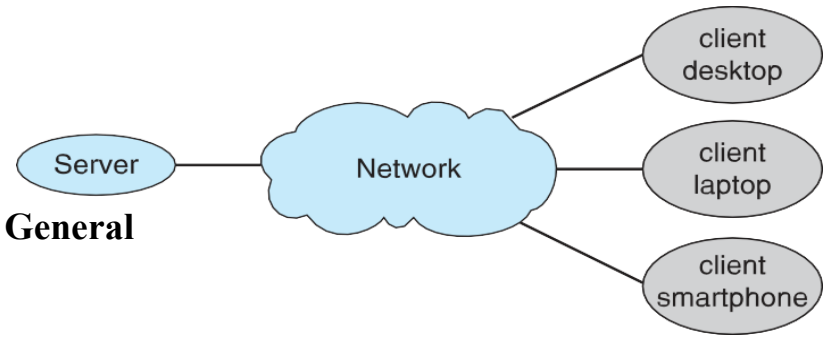
Distributed Systems

- Collection of separate, possibly heterogeneous, systems networked together
    - **Network** is a communications path, **TCP/IP** most common
        - **Local Area Network (LAN)**
        - **Wide Area Network (WAN)**
        - **Metropolitan Area Network (MAN)**
        - **Personal Area Network (PAN)**
- **Network Operating System** provides features between systems across network
    - Communication scheme allows systems to exchange messages
    - Different computers communicate closely enough to provide the Illusion that only single operating system controls the network.---- Distributed operating system

Client-Server Computing

Designers shifted away from centralized system architecture to - terminals connected to centralized systems. As a result, many of today's systems act as

server systems to satisfy requests generated by **client systems.** This form of specialized distributed system, called **client-server** system.



General

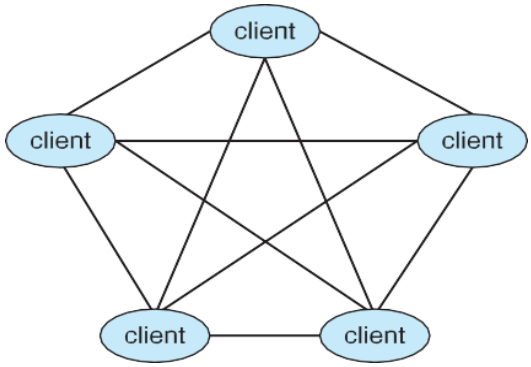
Structure of Client–Server System

Server systems can be broadly categorized as compute servers and files servers:

- The **compute-server system** provides an interface to which a client can send a request to perform an action (for example, read data); in response, the server executes the action and sends back results to the client. A server running a database that responds to client requests for data is an example of such a system.
- The **file-server system** provides a file-system interface where clients can create, update, read, and delete files. An example of such a system is a web server that delivers files to clients running the web browsers.

Peer-to-Peer Computing

- In this model, clients and servers are not distinguished from one another; here, all nodes within the system are considered peers, and each may act as either a client or a server, depending on whether it is requesting or providing a service.
- In a client-server system, the server is a bottleneck, because all the services must be served by the server. But in a peer-to-peer system, services can be provided by several nodes distributed throughout the network.
- To participate in a peer-to-peer system, a node must first join the network of peers. Once anode has joined the network, it can begin providing services to—and requesting services from—other nodes in the network.



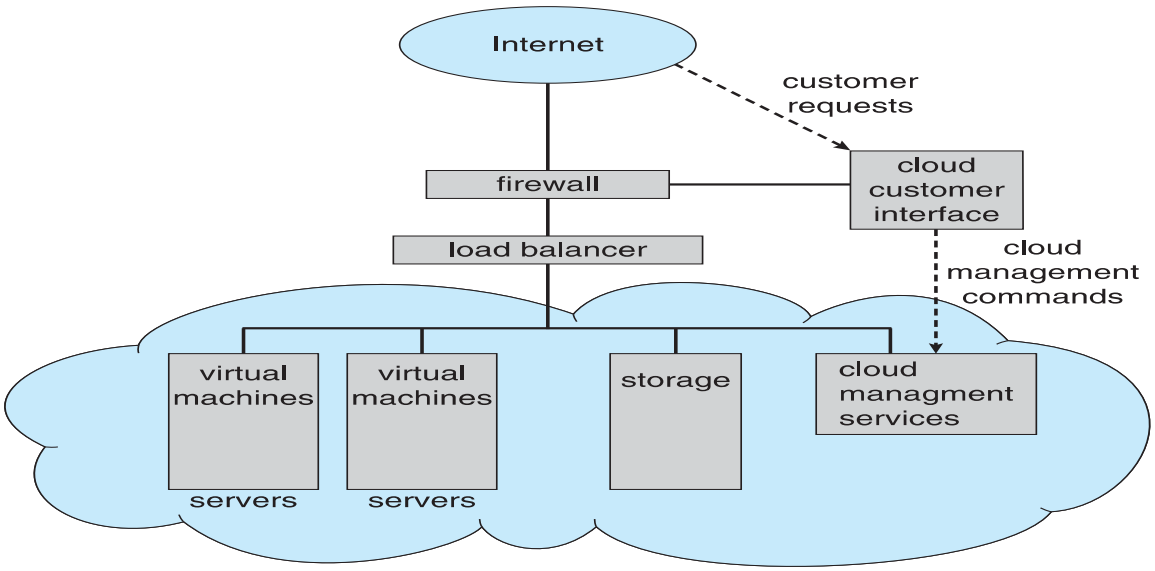
General Structure of P2P System

Ex : Skype is another example of peer-to-peer computing. It allows clients to make voice calls and video calls and to send text messages over the Internet using a technology known as voice over IP (VoIP). Skype uses a hybrid peer to-peer approach. It includes a centralized login server, but it also incorporates decentralized peers and allows two peers to communicate.

Cloud Computing

- Delivers computing, storage, even apps as a service across a network
- Logical extension of virtualization because it uses virtualization as the base for it functionality.
    - Amazon **EC2** has thousands of servers, millions of virtual machines, petabytes of storage available across the Internet, pay based on usage
- Many types
    - **Public cloud** – available via Internet to anyone willing to pay
    - **Private cloud** – run by a company for the company's own use
    - **Hybrid cloud** – includes both public and private cloud components
    - Software as a Service (**SaaS**) – one or more applications available via the Internet (i.e., word processor)
    - Platform as a Service (**PaaS**) – software stack ready for application use via the Internet (i.e., a database server)
    - Infrastructure as a Service (**IaaS**) – servers or storage available over Internet (i.e., storage available for backup use)
- Cloud computing environments composed of traditional OSes, plus VMMs, plus cloud management tools
    - Internet connectivity requires security like firewalls
    - Load balancers spread traffic across multiple applications

Cloud Computing Environment

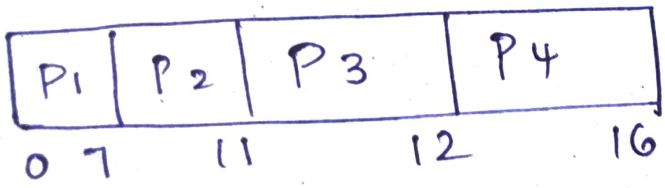


4) Given,

Process	AT (ms)	BT (ms)
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

(a) (i) FCFS

Gantt chart:



waiting time : $P_1 = 0$

$P_2 = 7-2 = 5$

$P_3 = 11-4 = 7$

$P_4 = 12-5 = 7$

Avg wt : $\frac{0+5+7+7}{4} = 4.75$

TAT: P_1 - 7

P_2 - 9

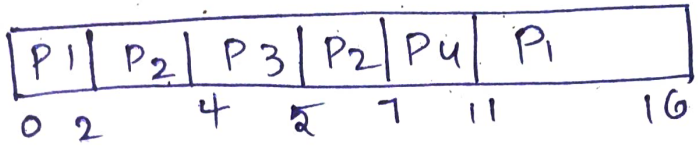
P_3 - 8

P_4 - 11

Avg TAT = 8.75

(2) SJF (~~Non~~-Preemptive)

Gantt chart



WT: P_1 - 9

P_2 - 1

P_3 - 0

P_4 - 2

Avg wt: 3ms

TAT: P_1 - 16

P_2 - 5

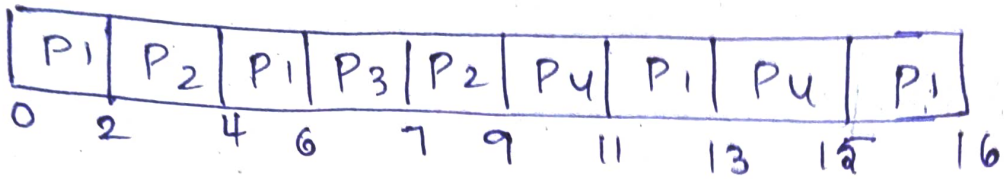
P_3 - 1

P_4 - 6

Avg TAT = 7ms

3) Round-Robin (TQ=2)

Gantt chart:



WT: P_1 - 9
P_2 - 3
P_3 - 2
P_4 - 6

Avg wt = 5ms

TAT: P_1 - 16
P_2 - 7
P_3 - 3
P_4 - 10

Avg ~~wt~~ (TAT): 9ms

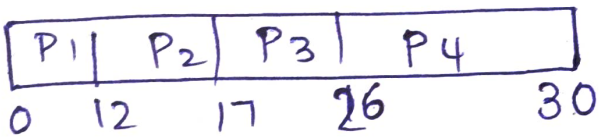
∴ SJF (preemptive) gives the minimum Avg wt (3ms) and TAT (7ms). However, it may cause starvation for long processes.

5) Given,

Process	AT(ms)	BT(ms)
P_1	0	12
P_2	1	5
P_3	2	9
P_4	3	4

(1) FCFS:

Gantt chart:



WT: P_1 - 0

P_2 - 11

P_3 - 15

P_4 - 23

Avg wt: 12.25

TAT: P_1 - 12

P_2 - 16

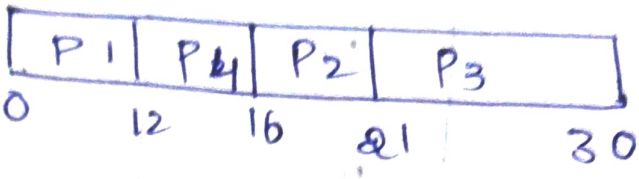
P_3 - 24

P_4 - 27

Avg TAT = 19.75

(2) SJF (Non-preemptive)

Gantt chart



WT: P1 - 0

P2 - 9

P3 - 15

P4 - 19

Avg wt: 10.75

TAT: P1 - 12

P2 - 13

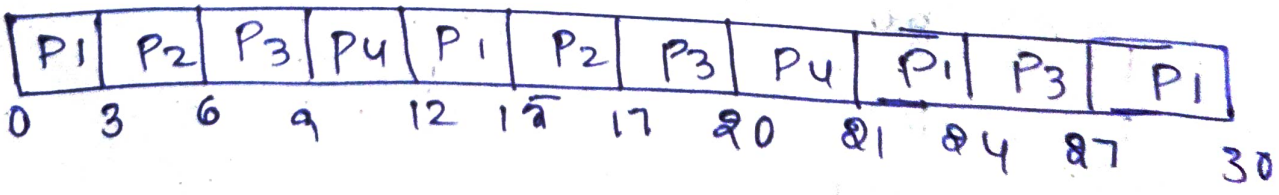
P3 - 20

P4 - 28

Avg TAT: 18.25

(3) Round-Robin (TQ=3)

Gantt chart:



WT: P_1 - 18

P_2 - 11

P_3 - 16

P_4 - 14

Avg wt: 14.75

TAT: P_1 - 30

P_2 - 16

P_3 - 25

P_4 - 18

Avg TAT: 22.25

∴ Hence, Among FCFS, SJF(NP) & RR, SJF (Non-preemptive) gives minimum avg wating time (10.75ms). It is most efficient.

UNIT III

Q6 (a) Explain Peterson's solution to the critical section problem with a suitable example. 5M

Peterson's solution is a classical software solution to the critical section problem for two processes. It ensures:

1. Mutual Exclusion
2. Progress
3. Bounded Waiting

It uses two shared variables to coordinate process execution.

Shared Variables

1. **flag[2]**
   - Indicates whether a process wants to enter the critical section.
   - flag[i] = true means process Pi wants to enter.
2. **turn**
   - Indicates whose turn it is to enter the critical section.

Algorithm for Process Pi

```
do {
    flag[i] = TRUE;
    turn = j;
    while (flag[j] && turn == j);

        critical section

    flag[i] = FALSE;

        remainder section

} while (TRUE);
```

Where:

- i= current process
- j= other process

Working Procedure

1. Process Pi sets flag[i] = true.
2. It gives chance to other process by setting turn = j.
3. If other process also wants to enter and it is its turn, Pi waits.
4. Once the other process finishes, Pi enters the critical section.
5. After completion, Pi sets flag[i] = false.

Advantages

1. Simple and efficient software solution
2. No hardware support required
3. Prevents race condition
4. Ensures fairness

Disadvantages

1. Applicable only for two processes
2. Busy waiting wastes CPU time

Q6 (b) Explain Banker's Algorithm for deadlock avoidance with a suitable example. 5M

Banker's Algorithm is a deadlock avoidance algorithm used in operating systems. It checks whether the requested resources can be allocated safely without causing deadlock. The request is granted only if the system remains in a safe state.

Data Structures Used

- **Available:**
  - A vector of length *m* indicates the number of available resources of each type.
  - If *Available[j]* equals *k,* then *k* instances of resource type *Ri* are available.
- **Max:**
  - An *n* x *m* matrix defines the maximum demand of each process.
  - If *Max[i] [j]* equals *k,* then process P_i may request at most *k* instances of resource type R_j.
- **Allocation:**
  - An *n* x *m* matrix defines the number of resources of each type currently allocated to each process.
  - If *Allocation[i][j]* equals k, then process P_i is currently allocated k instances of resource type R_j.
- **Need:**
  - An *n* x *m* matrix indicates the remaining resource need of each process.
  - If *Need[i][j]* equals *k,* then process P_i may need *k* more instances of resource type R_j to complete its task. Note that *Need[i][j]* equals *Max[i][j]- Allocation [i][j].*

Need = Max - Allocation

Safety Algorithm:

- Let *Work* and *Finish* be vectors of length *m* and *n,* respectively. Initialize
- *Work= Available* and *Finish[i] =false* for $i = 0, 1, \ldots, n - 1$.

2.Find an index *i* such that both

a. *Finish[i] ==false*

b. $Need_i <= Work$

- If no such *i* exists, go to step 4.

3. Work = Work + Allocation$_i$

Finish[i] = true

Go to step 2.

- 4. If *Finish[i] ==true* for all *i,* then the system is in a safe state.
- This algorithm may require an order of *m* x n^2 operations to determine whether a state is safe.

Resource-Request Algorithm

- This algorithm for determining whether requests can be safely granted.
- Let *Request$_i$* be the request vector for process P_i.
- If *Request$_i$ [j]==k,* then process P_i wants *k* instances of resource type R_j.

1. If *Request$_i$ <=Need$_i$,* go to step 2. Otherwise, raise an error condition, since the process has exceeded its maximum claim.

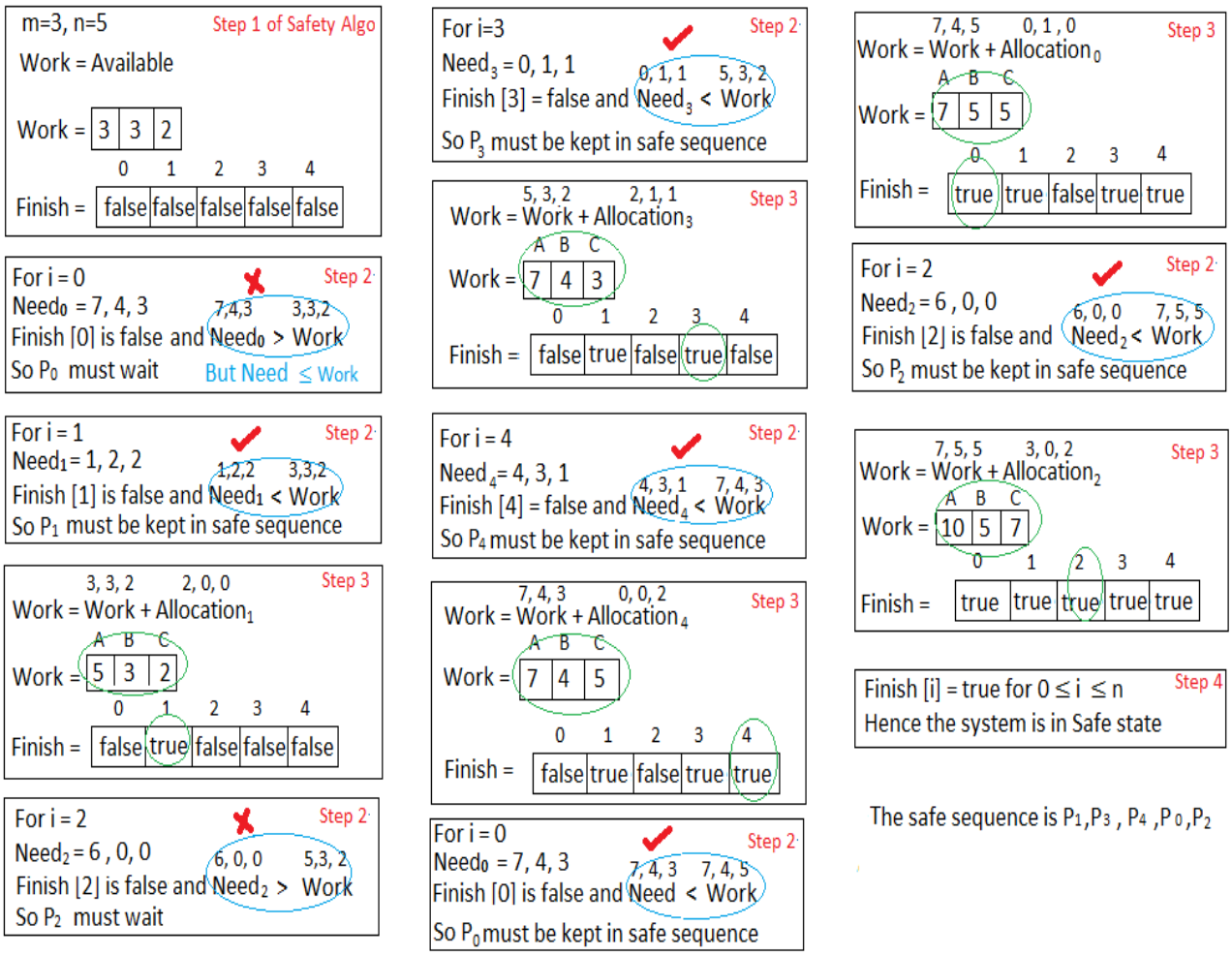
2. If *Request$_i$ <=Available,* go to step 3. Otherwise, P_i must wait, since the resources are not available.

3. Have the system pretend to have allocated the requested resources to process P_i by modifying the state as follows:

  - *Available= Available- Request$_i$;*
  - *Allocation$_i$ =Allocation$_i$ +Request$_i$;*
  - *Need$_i$ =Need$_i$- Request$_i$;*

- No of processes=5
- No of resources=3
- Instances of resources= A-10 B-5 C-7

AVAILABLE: 3 3 2			
Process	Allocation	Max	Need
	A B C	A B C	A B C
P0	0 1 0	7 5 3	7 4 3
P1	2 0 0	3 2 2	1 2 2
P2	3 0 2	9 0 2	6 0 0
P3	2 1 1	2 2 2	0 1 1
P4	0 0 2	4 3 3	4 3 1



- The system is in a safe state since the sequence < P_1, P_3, P_4, P_2, P_0> satisfies safety criteria.

Q7 (a) Describe Semaphores and their usage. Explain how semaphores are used to solve Producer-Consumer synchronization problem. 5M

Semaphore is a synchronization tool used in operating systems to control access to shared resources. It helps processes coordinate their activities and avoids race conditions.

A semaphore is an integer variable that can be accessed only through two atomic operations:

1. wait()
2. signal()

Types of Semaphores

1. Binary Semaphore
   Value can be 0 or 1. Used for mutual exclusion.
2. Counting Semaphore

Value can be any non-negative integer. Used when multiple resources exist.

Producer-Consumer Problem:

- We assume that the pool consists of *n* buffers, each capable of holding one item.
- The mutex semaphore provides mutual exclusion for accesses to the buffer pool and is initialized to the value 1.
- The empty and full semaphores count the number of empty and full buffers.
- The semaphore empty is initialized to the value n; the semaphore full is initialized to the value 0.

The code for the producer process is shown below

```
do {
    . . .
   // produce an item in nextp
    . . .
   wait(empty);
   wait(mutex);
    . . .
   // add nextp to buffer
    . . .
   signal(mutex);
   signal(full);
} while (TRUE);
```

The code for the consumer process is shown below:

```
do {
  wait(full);
  wait(mutex);
    . . .
  // remove an item from buffer to nextc
    . . .
  signal(mutex);
  signal(empty);
    . . .
  // consume the item in nextc
    . . .
} while (TRUE);
```

- The producer producing full buffers for the consumer or as the consumer producing empty buffers for the producer.

Q7(b) Explain how semaphores are used to solve Readers-Writers problem. 5M

- Suppose that a database is to be shared among several concurrent processes.
- Some of these processes may want only to read the database, whereas others may want to update the database.

- These two types of processes are distinguished as readers and writers
- If two readers access the shared data simultaneously, no adverse effects will result.
- If a writer and some other process (either a reader or a writer) access the database then problem arises.
- To ensure that these difficulties do not arise, we require that the writers have exclusive access to the shared database while writing to the database.
- This synchronization problem is referred to as the *readers-writers problem.*
- The readers-writers problem has several variations
- The *first readers-writers problem:*
- It requires that no reader be kept waiting unless a writer has already obtained permission to use the shared object.
- In other words, no reader should wait for other readers to finish simply because a writer is waiting.
- The *second readers writers* problem:
- It requires that, once a writer is ready, that writer performs its write as soon as possible.
- In other words, if a writer is waiting to access the object, no new readers may start reading.
- A solution to either problem may result in starvation. In the first case, writers may starve; in the second case, readers may starve.
- In the solution to the first readers-writers problem, the reader processes share the following data structures:

```
semaphore mutex, wrt;
int readcount;
```

-
- The semaphores mutex and wrt are initialized to 1;
- readcount is initialized to 0.
- The semaphore wrt is common to both reader and writer processes.
- The mutex semaphore is used to ensure mutual exclusion when the variable readcount is updated.
- The readcount variable keeps track of how many processes are currently reading the object.
- The semaphore wrt functions as a mutual-exclusion semaphore for the writers.

  The structure of a reader process

```
do {
   wait(mutex);
   readcount++;
   if (readcount == 1)
      wait(wrt);
   signal(mutex);
      . . .
   // reading is performed
      . . .
   wait(mutex);
   readcount--;
   if (readcount == 0)
      signal(wrt);
   signal(mutex);
} while (TRUE);
```

The structure of a writer process

```
do {
  wait(wrt);
     . . .
  // writing is performed
     . . .
  signal(wrt);
} while (TRUE);
```

UNIT IV

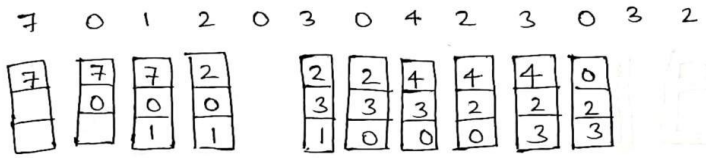
8a) Apply FIFO and Optimal Page Replacement algorithms to the following page reference string with three frames: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 Calculate the number of page faults and identify which algorithm performs better.

8.a) Given Page Reference String:

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

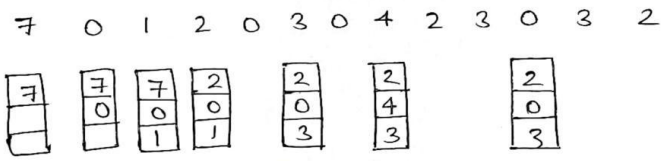
No. of frames = 3

FIFO(First in First out):



Total FIFO page faults = 10

Optimal Page Replacement:



Total optimal page replacement faults = 7

∴ Optimal page replacement performs better

8b) Consider the disk request queue:
176, 79, 34, 60, 92, 11, 41, 114 Initial head position = 50 Disk cylinder range = 0-199
The disk head is initially moved toward high-numbered cylinders. Compute the total head movement for the following disk scheduling algorithms 5M

i) SCAN ii) C-SCAN

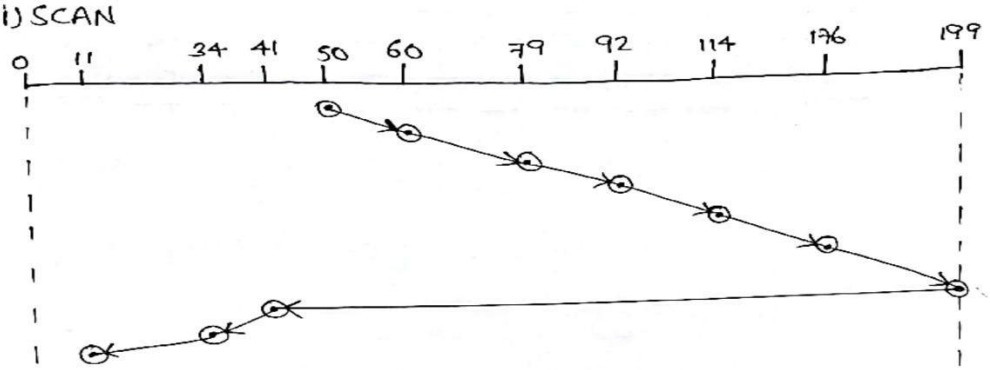
8.b) Queue:

176, 79, 34, 60, 92, 11, 41, 114

Initial Head = 50

Direction ⇒ towards higher cylinders

(i) SCAN

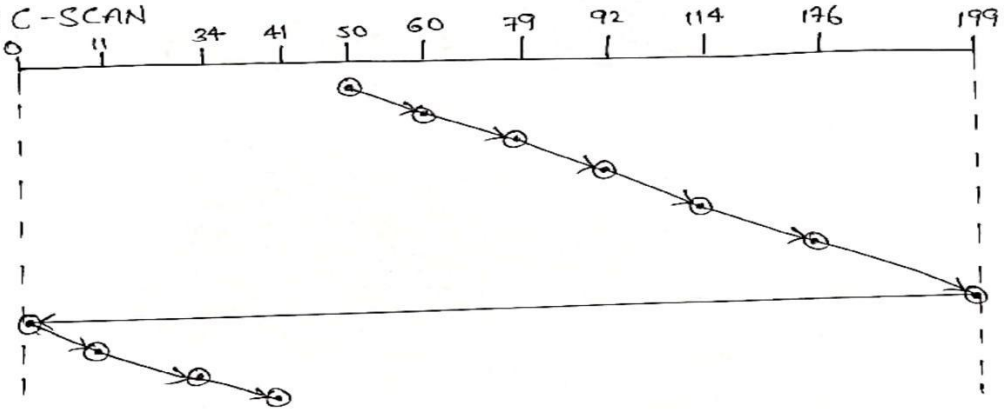


Total head movements =

|50-60| + |60-79| + |79-92| + |92-114| + |114-176| + |176-199| +

|199-41| + |41-34| + |34-11| = 337

(ii) C-SCAN



Total head movements = |50-60| + |60-79| + |79-92| + ~~|114-15|~~

|114-92| + |114-176| + |176-199| + |199-0| + |0-11| +

|11-34| + |34-41| = 389

9a) Consider the page reference string below with four frames:
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5 Apply LRU and FIFO page replacement algorithms and compute page faults. Compare results. 5M

FIFO: Page faults: 10

Step	Page	Frames	Result
1	1	1 - - -	F
2	2	1 2 - -	F
3	3	1 2 3 -	F
4	4	1 2 3 4	F
5	1	1 2 3 4	H
6	2	1 2 3 4	H
7	5	5 2 3 4	F (1 removed)
8	1	5 1 3 4	F (2 removed)
9	2	5 1 2 4	F (3 removed)
10	3	5 1 2 3	F (4 removed)
11	4	4 1 2 3	F (5 removed)
12	5	4 5 2 3	F (1 removed)

LRU No of page faults:8

Step	Page	Frames	Result
1	1	1 - - -	F
2	2	1 2 - -	F
3	3	1 2 3 -	F
4	4	1 2 3 4	F
5	1	1 2 3 4	H
6	2	1 2 3 4	H
7	5	1 2 5 4	F
8	1	1 2 5 4	H
9	2	1 2 5 4	H
10	3	1 2 5 3	F
11	4	1 2 4 3	F
12	5	5 2 4 3	F

Conclusion: LRU better than FIFO

9b) Given five processes and 20 available frames, allocate frames using: i) Equal Allocation ii) Proportional allocation Process sizes (pages): P1=10, P2=20, P3=30, P4=25, P5=15 5M

Determine number of assigned to each process.

q.b) Total no.of frames = 20

Processes:

$P_1 = 10$

$P_2 = 20$

$P_3 = 30$

$P_4 = 25$

$P_5 = 15$

Total pages =

$10+20+30+25+15 = 100$

(i) Equi allocation:

Frames per process = $\frac{\text{Total frames}}{\text{No.of processes}}$

$= \frac{20}{5} = 4$ frames each

$P_1 = 4,\ P_2 = 4,\ P_3 = 4,\ P_4 = 4,\ P_5 = 4$

(ii) Proportional Allocation:

Frames = (Process size / Total pages) x Total frames

$P_1 = \frac{10}{100} \times 20 = 2$

$P_2 = \frac{20}{100} \times 20 = 4$

$P_3 = \frac{30}{100} \times 20 = 6$

$P_4 = \frac{25}{100} \times 20 = 5$

$P_5 = \frac{15}{100} \times 20 = 3$

UNIT-V

10 a) Explain the file concept in operating systems. Describe file attributes, file operations, and file types with examples 5M

- A file is a collection of related information that is recorded on secondary storage.
- A file is a smallest allotment of logical secondary storage that is data can't be written to secondary storage unless they are within a file.

File Attributes:

- **Name:** The symbolic file name is the only information kept in human readable code.
- **Identifier**: This is unique tag, usually a number, identifies the file within the file system. It is the non human readable name for the file.
- **Type**: This information is needed for those systems that support different type.
- **Location:** This information is a pointer to a device and to the location of the file on that device.
- **Size:** the current size of the file and possibly the maximum allowed size are included in this attribute.
- **Protection:** Access-control information determines who can do reading, writing, executing and so on.
- **Time & Date:** This information may be kept for creation, last modification, and last use. These data can be useful for protection, security and usage monitoring.

File Operations:

A file is an abstract data type. Operating system must do for each of the six basic file operations.

- **Creating a file:** Two steps are required to create a file. First, space in the file system must be found for the file. Second, an entry for the new file must be made in the directory.
- **Writing a file:** to write a file, we make a system call specifying both the name of the file and the information to be written to the file.
- **Reading a file:** to read from a file, we use a system call that specifies the name of the file and where the next block of the file should be put. Again, the directory is searched for the associated directory entry, and the system needs to keep a read pointer to the location in the file where the next read is take place. Once the read has taken place, the read pointer is updated.

- **Repositioning with in a file:** The directory is searched for the appropriate entry, and the current file position is set to a given value.
- **Deleting a file:** whatever files want to delete from directory, those files have to found in directory. If file is found then it will be deleted. Deleted file space used for store the next file.
- **Truncating a file:** The file attributes are not changed but the content of file deleted.
- There are several issues are associated with an open file.

❖ **File pointer:** it is a unique pointer for each process operating on file.

❖ **File open count:** how many times has the current file has been opened and not yet closed. When this counter reaches zero the file can be removed from the table.

❖ **Disk location of the file:** most file operations required to modify data within a file. The information needed to locate the file on disk is kept in memory so that the system does not have to read it from disk for each operation.

❖ **Access right:** Each process can open a file in access mode. This information stored on pre process table. So that the operating system can allow or deny subsequent I/O operations.

File Types and Examples

The type of a file is usually indicated by its extension, which helps the operating system recognize its internal structure and intended function. Common examples include:

file type	usual extension	function
executable	exe, com, bin or none	read to run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rrf, doc	various word-processor formats
library	lib, a, so, dll, mpeg, mov, rm	libraries of routines for programmers
print or view	arc, zip, tar	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm	binary file containing audio or A/V information

10 b) Explain free space management techniques in file systems. Compare bit vector and linked list methods 5M

Free space management is the process by which an operating system tracks and manages unused disk blocks to ensure efficient storage utilization. The system maintains a free-space list, which records all blocks not currently allocated to files. When a new file is created, the system searches this list for the required space, allocates it, and removes those blocks from the list; conversely, when a file is deleted, its blocks are returned to the free-space list.

Free Space Management Techniques

The sources identify two primary techniques for implementing a free-space list:

1. Bit Vector (or Bit Map)

In this approach, the free-space list is implemented as a map where **each disk block is represented by a single bit**.

- **Representation:** A bit value of **1 indicates the block is free**, while a **0 indicates it is allocated**.
- **Efficiency:** This method is relatively simple and highly efficient for **finding the first free block** or locating a sequence of **consecutive free blocks**.
- **Calculation:** The specific block number can be calculated using the formula: (number of bits per word) × (number of 0-valued words) + offset of first 1 bit.
- **Space Requirement:** A disadvantage is that the bit map itself **requires extra memory space**. For example, a 1-terabyte disk with 4KB blocks would require approximately 32MB of memory to store the bit map.

2. Linked List

This technique **links together all free disk blocks**, maintaining a pointer to the first free block in a special location on the disk and caching it in memory.

- **Mechanism:** The first free block contains a pointer to the next free block, which in turn points to the next, creating a continuous chain of all available space.
- **Inefficiency:** This scheme is generally **not efficient for traversal**. To find multiple free blocks, the system must read each block in the chain, which requires **substantial I/O time**.

Comparison of Bit Vector and Linked List

Feature	Bit Vector (Bit Map)	Linked List
Search Efficiency	High; very fast at finding individual or contiguous free blocks.	Low; requires sequential I/O to traverse the list.
Implementation	Uses a bit array (1 for free, 0 for allocated).	Uses a chain of pointers stored within the free blocks themselves.
Space Overhead	Requires additional memory to store the map (proportional to disk size).	Minimal overhead, as pointers are stored in the free blocks themselves.
Best Use Case	Systems needing fast allocation and contiguous space.	Systems where memory for a bit map is extremely limited.

11 a) Discuss different directory structures used in file systems. Compare single-level, two-level, and tree-structured directories. 5M

Single-Level Directory

- The simplest directory structure is the single-level directory
- All files are contained in single directory which is easy to support and understand.
- It has a limitation when the number of files increases or when the system has more than one user.
- Since all files are in the same directory, they must have unique names.
- In single level directory all file names are unique.
- If two users call their data file name as test, then unique-name rule is violated.
- Even a single user on a single-level directory may find it difficult to remember the names of all the files as the number of files increases.

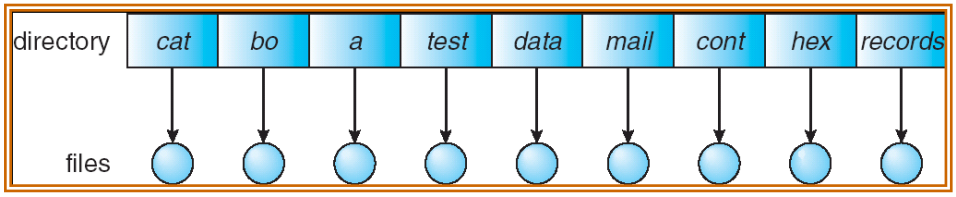


Fig: Single-level directory.

Two-Level Directory

- A single-level directory often leads to confusion of file names among different users.
- The standard solution is to create a *separate* directory for each user.
- In the two-level directory structure, each user has his own UFD
- The UFDs have similar structures, but each lists only the files of a single user.
- When a user job starts or a user logs in, the system's MFD is searched.
- The MFD is indexed by user name or account number, and each entry points to the UFD for that user
- When a user refers to a particular file, only his own UFD is searched.
- Thus, different users may have files with the same name, as long as all the file names within each UFD are unique.
- To create a file for a user, the operating system searches only that user's UFD to ascertain whether another file of that name exists.
- To delete a file, the operating system confines its search to the local UFD; thus, it cannot accidentally delete another user's file that has the same name.
- The user directories themselves must be created and deleted as necessary.
- A special system program is run with the appropriate user name and account information.
- The program creates a new UFD and adds an entry for it to the MFD.
- The execution of this program might be restricted to system administrators.
- Although the two-level directory structure solves the name-collision problem, it still has disadvantages.
- This structure effectively isolates one user from another.
- Isolation is an advantage when the users are completely independent but is a disadvantage when the users *want* to cooperate on some task and to access one another's files.
- Some systems simply do not allow local user files to be accessed by other users.

- If access is to be permitted, one user must have the ability to name a file in another user's directory.
- To name a particular file "UNIQUELY in a two-level directory, we must give both the user name and the file name.
- A two-level directory can be thought of as a tree, or an inverted tree, of height 2.
- The root of the tree is the MFD. Its direct descendants are the UFDs. The descendants of the UFDs are the files themselves. The files are the leaves of the tree.
- Specifying a user name and a file name defines a path in the tree from the root (the MFD) to a leaf (the specified file). Thus, a user name and a file name define a *path name.*
- Every file in the system has a path name.
- To name a file uniquely, a user must know the path name of the file desired.
- For example, if user A wishes to access her own test file named *test,* she can simply refer to *test.*
- To access the file named *test* of user B (with directory-entry name *userb),* however, she might have to refer to */userb/test.*
- Additional syntax is needed to specify the volume of a file. a file specification might be *C:\userb\fest.*
- As the directory system is defined presently, this file name would be searched for in the current UFD.
- One solution would be to copy the system files into each UFD.
- However, copying all the system files would waste an enormous amount of space.
- The standard solution is a special user directory is defined to contain the system files.
- Whenever a file name is given to be loaded, the operating system first

searches the local UFD. If the file is found, it is used.

- If it is not found, the system. It automatically searches the special user directory that contains the system files.

- The sequence of directories searched when a file is named is called the search path

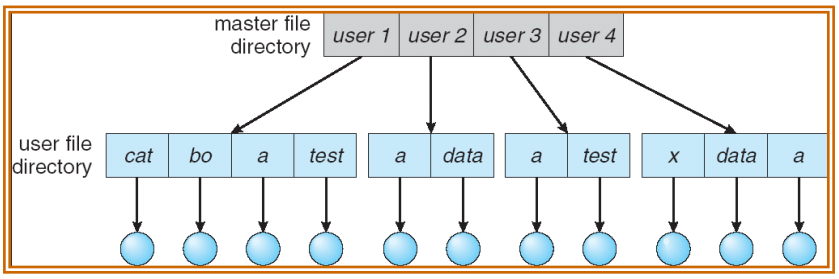


Fig: Two-level Directory Structure

Tree-Structured Directories

- Once we have seen how to view a two-level directory as a two-level tree,
- The natural generalization is to extend the directory structure to a tree of arbitrary height.
- This generalization allows users to create their own subdirectories and to organize their files accordingly.
- A tree is the most common directory structure.
- The tree has a root directory, and every file in the system has a unique path name.
- A directory (or subdirectory) contains a set of files or subdirectories.
- All directories have the same internal format.
- One bit in each directory entry defines the entry as a file (0) or as a subdirectory (1).
- Special system calls are used to create and delete directories.
- Path names can be of two types: *absolute* and *relative.*
- An *absolute path* begins at the root and follows a down to the specified file, giving the directory names on the path.
- A relative path defines a path from the current directory.
- An interesting policy decision in a tree-structured directory concerns how to handle the deletion of a directory.
- If a directory is empty, its entry in the directory that contains it can simply be deleted.

- However, suppose the directory to be deleted is not empty but contains several files or subdirectories. One of two approaches can be taken.
- With a tree-structured directory system, users can be allowed to access, in addition to their files, the files of other users.
- A path to a file in a tree-structured directory can be longer than a path in a two-level directory.
- To allow users to access programs without having to remember these long paths, the Macintosh operating system automates the search for executable programs.

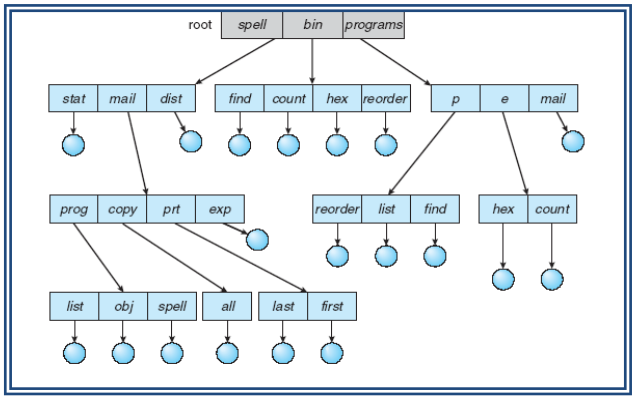


Fig: Tree-structured directory structure

Acyclic-Graph Directories

- Consider two programmers who are working on a joint project.
- The files associated with that project can be stored in a subdirectory, separating them from other projects and files of the two programmers.
- But since both programmers are equally responsible for the project, both want the subdirectory to be in their own directories.
- The common subdirectory should be *shared.*
- A shared directory or file will exist in the file system in two (or more) places at once.
- A tree structure prohibits the sharing of files or directories.
- An acyclic graph-that is, a graph with no cycles- *It* allows directories to share subdirectories and files
- The *same* file or subdirectory may be in two different directories.

- The acyclic graph is a natural generalization of the tree-structured directory scheme.
- It is important to note that a shared file (or directory) is not the same as two copies of the file.
- With two copies, each programmer can view the copy rather than the original, but if one programmer changes the file, the changes will not appear in the other's copy.
- With a shared file, only *one* actual file exists, so any changes made by one person are immediately visible to the other.
- Sharing is particularly important for subdirectories; a new file created by one person will automatically appear in all the shared subdirectories.

Implementation of shared files

1. **Link:**
2. **Duplication**

- An acyclic-graph directory structure is more flexible than is a simple tree structure, but it is also more complex.
- Several problems must be considered carefully.
- A file may now have multiple absolute path names.
- Consequently, distinct file names may refer to the same file.
- This situation is similar to the aliasing problem for programming languages.
- Another problem involves deletion.
- When can the space allocated to a shared file be deallocated and reused?

1. **Using links**
2. **Reference list**
3. **Reference count**

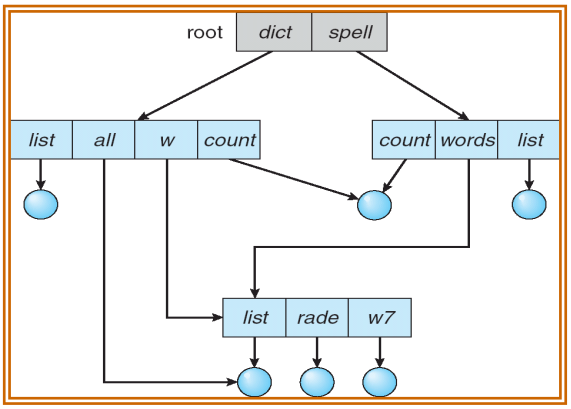


Fig: Acyclic-graph directory structure

General Graph Directory

- A serious problem with using an acyclic-graph structure is ensuring that there are no cycles.
- We want to avoid traversing shared sections of an acyclic graph twice, mainly for performance reasons.
- When we are searching a file, search operation is going into an infinite loop in cyclic structure.
- A similar problem exists when we are trying to determine when a file can be deleted.
- With acyclic-graph directory structures, a value of 0 in the reference count means that there are no more references to the file or directory, and the file can be deleted.
- However, when cycles exist, the reference count may not be 0 even when it is no longer possible to refer to a directory or file.
- This anomaly results from the possibility of self-referencing (or a cycle) in the directory structure.
- In this case, we generally need to use a garbage-collection scheme to determine when the last reference has been deleted and the disk space can be reallocated.
- Garbage collection involves traversing the entire file system, marking everything that can be accessed.
- Then, a second pass collects everything that is not marked onto a list of free space
- Garbage collection for a disk-based file system, however, is extremely time consuming.

- Garbage collection is necessary only because of possible cycles in the graph.
- Thus, an acyclic-graph structure is much easier to work with.
- There are algorithms to detect cycles in graphs; however, they are computationally expensive, especially when the graph is on disk storage.
- A simpler algorithm in the special case of directories and links is to bypass links during directory traversal.
- Cycles are avoided, and no extra overhead is incurred.

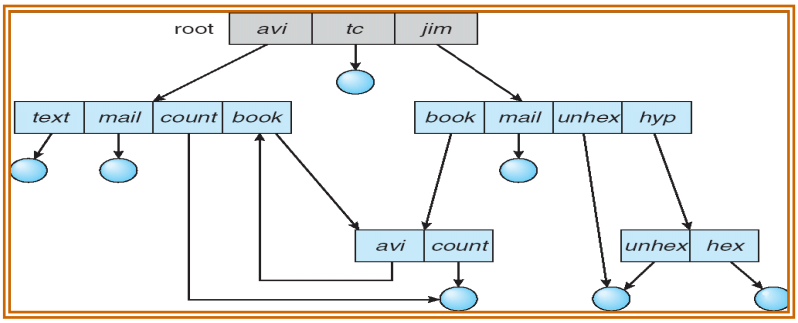


Fig: General graph directory

Feature	**Single-Level Directory**	**Two-Level Directory**	**Tree-Structured Directory**
Structure	Single directory	MFD + UFD (per user)	Hierarchical (multi-level)
File Naming	Global uniqueness required	Unique within each user	Unique within each directory
User Separation	Not supported	Supported	Supported
Scalability / Organization	Poor	Moderate	Excellent

11 b) Explain protection mechanisms in operating systems. Describe protection goals, protection domains, and access matrix with an example 5M

Protection mechanisms in operating systems are designed to manage and enforce policies that govern how resources are used by different processes.

Goals of Protection

The primary objectives of implementing protection in an operating system include:

- **Preventing Violations:** Protecting the system from intentional, mischievous violations of access restrictions by users.
- **Improving Reliability:** Enhancing system reliability by detecting "latent errors" at the interfaces between different component subsystems.
- **Distinguishing Usage**: Providing a clear means to distinguish between authorized and unauthorized usage of resources.
- **Enforcement of Policies:** Providing a flexible mechanism to enforce various policies regarding resource use, whether established by system design or management.

Protection Domains

A computer system consists of processes and objects, which include hardware (CPU, memory, disks) and software (files, programs, semaphores). Protection is implemented through protection domains:

- **Need-to-Know Principle:** A process should only be able to access the specific resources it currently requires to complete its task.
- **Domain Structure:** A protection domain defines a set of objects and the types of operations allowed on them.
- **Access Rights:** The ability to perform an operation on an object is an "access right," represented as an ordered pair: <object-name, rights-set>.
- **Realization:** Domains can be associated with individual users, processes, or procedures.

The Access Matrix

The Access Matrix is an abstract model used to view protection. It is structured as follows:

- **Rows:** Represent the various protection domains.
- **Columns:** Represent the objects within the system.
- **Entries:** The entry access (i, j) specifies the set of operations (such as read, write, or execute) that a process executing in domain can perform on object.

The access matrix design separates mechanism (the rules the OS strictly enforces) from policy (the user's decision on who can access what). It can also include special

rights like switch (allowing a process to move between domains), copy (*) (allowing rights to be shared), and owner (allowing a process to add or remove rights in a column).

Example of an Access Matrix

Consider a system with four domains (through) and four objects: three files () and a laser printer.

Domain	File F1	File F2	File F3	Laser Printer
D1	**Read**		**read**	
D2				**print**
D3		**read**	**execute**	
D4	**read, write**		**read, write**	

In this example:

- Processes in domain D1 can read files F1 and F3.
- The laser printer is exclusive to domain D2; only processes in this domain can use the print operation.
- Domain D4 has expanded privileges compared to D1, as it can both read and write files F1 and F3.