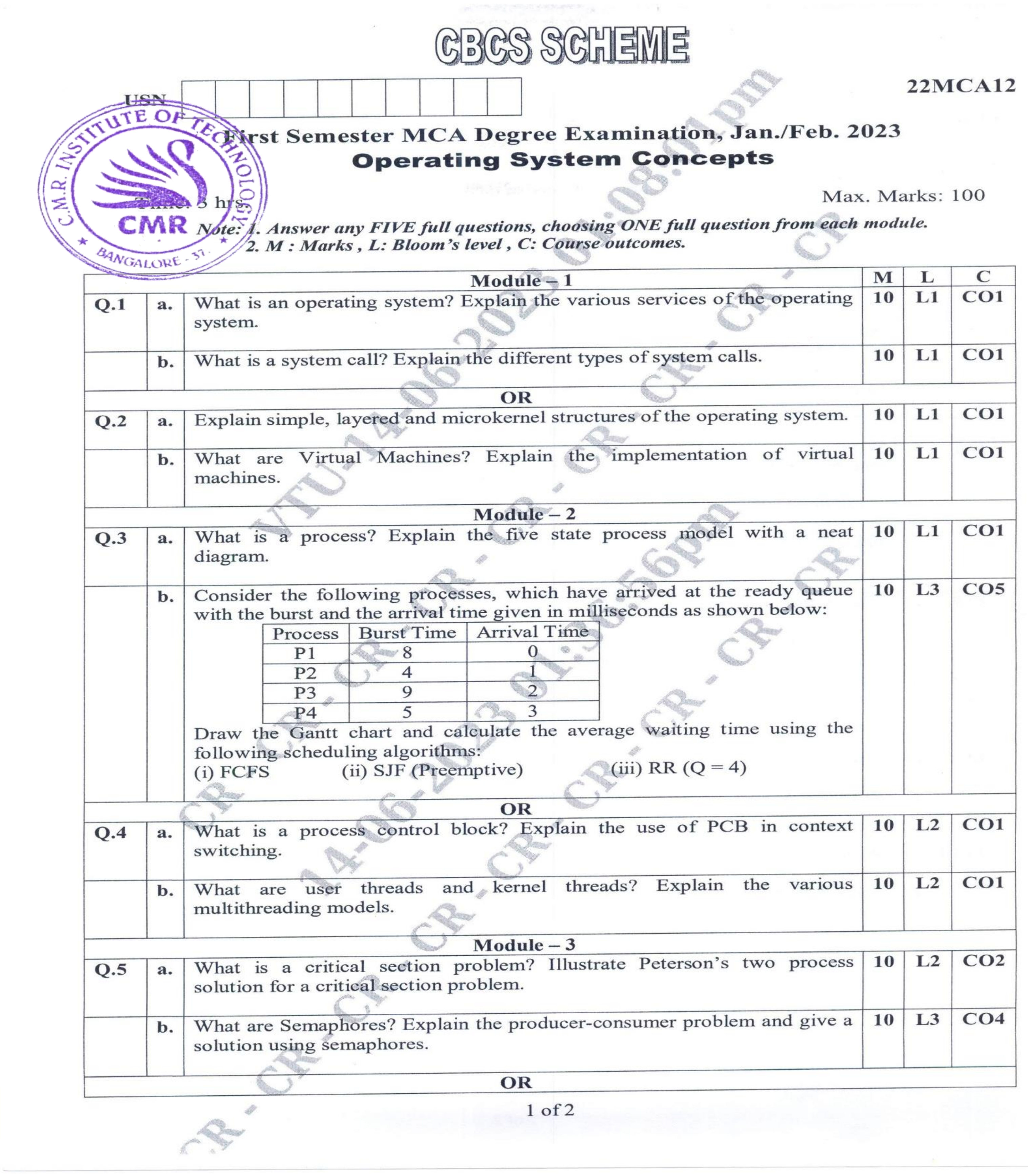
Sub:	VTU QP Solutions Operating System Concepts						
Sub code	22MCA12	Duration:	3 hrs	Max Marks:	100	Sem:	I
Note : Answer FIVE FULL Questions, choosing ONE full question from each Module							



CBCS SCHEME

USN

22MCA12

First Semester MCA Degree Examination, Jan./Feb. 2023

Operating System Concepts

Time: 3 hrs. Max. Marks: 100

Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.
2. M : Marks , L: Bloom's level , C: Course outcomes.

			M	L	C
		Module – 1			
Q.1	a.	What is an operating system? Explain the various services of the operating system.	10	L1	CO1
	b.	What is a system call? Explain the different types of system calls.	10	L1	CO1
		OR			
Q.2	a.	Explain simple, layered and microkernel structures of the operating system.	10	L1	CO1
	b.	What are Virtual Machines? Explain the implementation of virtual machines.	10	L1	CO1
		Module – 2			
Q.3	a.	What is a process? Explain the five state process model with a neat diagram.	10	L1	CO1
	b.	Consider the following processes, which have arrived at the ready queue with the burst and the arrival time given in milliseconds as shown below: Process: P1, P2, P3, P4; Burst Time: 8, 4, 9, 5; Arrival Time: 0, 1, 2, 3 Draw the Gantt chart and calculate the average waiting time using the following scheduling algorithms: (i) FCFS (ii) SJF (Preemptive) (iii) RR (Q = 4)	10	L3	CO5
		OR			
Q.4	a.	What is a process control block? Explain the use of PCB in context switching.	10	L2	CO1
	b.	What are user threads and kernel threads? Explain the various multithreading models.	10	L2	CO1
		Module – 3			
Q.5	a.	What is a critical section problem? Illustrate Peterson's two process solution for a critical section problem.	10	L2	CO2
	b.	What are Semaphores? Explain the producer-consumer problem and give a solution using semaphores.	10	L3	CO4
		OR			

Process	Burst Time	Arrival Time
P1	8	0
P2	4	1
P3	9	2
P4	5	3

C.M.R. INSTITUTE OF TECHNOLOGY
CMR
BANGALORE - 37.

Q.6	**a.**	What is a deadlock? What are the necessary conditions for a deadlock to occur?	**10**	**L2**	**CO2**
	b.	Consider the following snapshot of a system:	**10**	**L3**	**CO5**

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	3	3	2
P_1	2	0	0	3	2	2			
P_2	3	0	2	9	0	2			
P_3	2	1	1	2	2	2			
P_4	0	0	2	4	3	3			

Answer the following questions using Bankers Algorithm:
(i) Construct a need matrix.
(ii) Is the system in a safe state? If yes, what is the safe sequence?
(iii) If process P_1 makes a request (1, 0, 2), can the request be granted?

		Module – 4			
Q.7	**a.**	Write a C program to simulate the multiprogramming with variable member of tasks (MVT) memory management technique. Given the size of the memory, size of OS, number of processes and size of each process, calculate the external fragmentation.	**10**	**L3**	**CO5**
	b.	What is Paging? Explain the paging hardware with a neat diagram.	**10**	**L2**	**CO5**
		OR			
Q.8	**a.**	What is demand paging? Explain how demand paging can be implemented.	**10**	**L2**	**CO5**
	b.	Consider the following reference string:7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1 How many page faults would occur in case of the following page replacement algorithms: (i) Optimal (ii) LRV? Assuming 3 frames. Note : Initially all frames are empty.	**10**	**L3**	**CO3**
		Module – 5			
Q.9	**a.**	What are the various access methods used for accessing files?	**10**	**L2**	**CO5**
	b.	Explain the various directory structures with neat diagrams.	**10**	**L2**	**CO5**
		OR			
Q.10	**a.**	Write a note on different file allocation methods.	**10**	**L2**	**CO5**
	b.	Show how free space management is done using: (i) Bit vector (ii) Linked list (iii) Grouping (iv) Counting	**10**	**L3**	**CO5**

* * * * *

Module 1

Q1.a.What is an Operating system? Explain the various services of the operating system?
Operating system:

An operating system (OS) is a software program that acts as an intermediary between a user and their computer's hardware, allowing users to interact with and use their devices.

Batch operating system

The users of a batch operating system do not interact with the computer directly. Each user prepares his job on an off-line device like punch cards and submits it to the computer operator. To speed up processing, jobs with similar needs are batched together and run as a group. The programmers leave their programs with the operator and the operator then sorts the programs with similar requirements into batches.

The problems with Batch Systems are as follows −

- Lack of interaction between the user and the job.
- CPU is often idle, because the speed of the mechanical I/O devices is slower than the CPU.
- Difficult to provide the desired priority.

Time-sharing operating systems

Time-sharing is a technique which enables many people, located at various terminals, to use a particular computer system at the same time. Time-sharing or multitasking is a logical extension of multiprogramming. Processor's time which is shared among multiple users simultaneously is termed as time-sharing.

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.

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.

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.

Network operating System

A Network Operating System runs on a server and provides the server the capability to manage data, users, groups, security, applications, and other networking functions. The primary purpose of the network operating system is to allow shared file and printer access among multiple computers in a network, typically a local area network (LAN), a private network or to other networks.

Examples of network operating systems include Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux, Mac OS X, Novell NetWare, and BSD.

The advantages of network operating systems are as follows −

- Centralized servers are highly stable.
- Security is server managed.
- Upgrades to new technologies and hardware can be easily integrated into the system.
- Remote access to servers is possible from different locations and types of systems.

The disadvantages of network operating systems are as follows −

- High cost of buying and running a server.
- Dependency on a central location for most operations.
- Regular maintenance and updates are required.

Real Time operating System

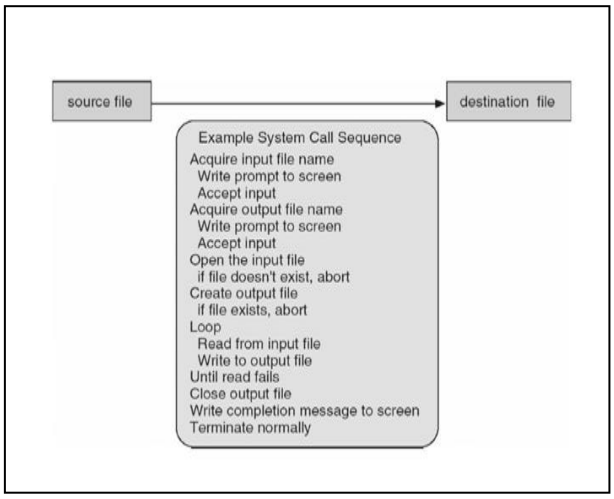
A real-time system is defined as a data processing system in which the time interval required to process and respond to inputs is so small that it controls the environment. The time taken by the system to respond to an input and display of required updated information is termed as the **response time**. 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.

Q1. b . What is System call? Explain the different types of system calls?

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

Example of System Calls

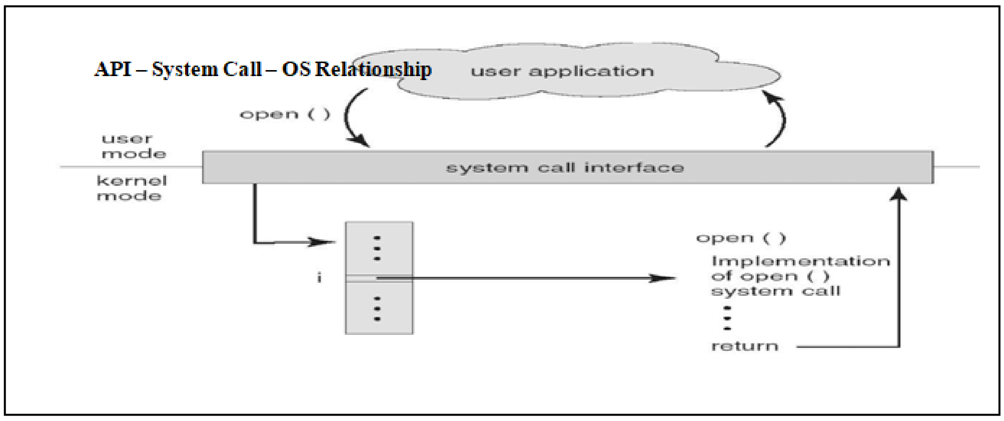


```
Acquire input file name
  Write prompt to screen
  Accept input
Acquire output file name
  Write prompt to screen
  Accept input
Open the input file
  if file doesn't exist, abort
Create output file
  if file exists, abort
Loop
  Read from input file
  Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally
```

System Call Implementation

- Typically, a number associated with each system call
- System-call interface maintains a table indexed according to these numbers
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about how the system call is implemented
- Just needs to obey API and understand what OS will do as a result call
- Most details of OS interface hidden from programmer by API Managed by run-time support library (set of functions built into libraries included with compiler)
-

API – System Call – OS Relationship



System Call Parameter Passing

- Often, more information is required than simply identity of desired system call
- Exact type and amount of information vary according to OS and call
- Three general methods used to pass parameters to the OS
- Simplest: pass the parameters in *registers*
    - In some cases, may be more parameters than registers
- Parameters stored in a *block*, or table, in memory, and address of block passed as a parameter in a register

This approach taken by Linux and Solaris

- Parameters placed, or *pushed*, onto the *stack* by the program and *popped* off the stack by the operating system
- Block and stack methods do not limit the number or length of parameters being passed

1.2.2 Types of System Calls

- Process control
- File management
- Device management
- Information maintenance
- Communications
- Protection

Process Control

A running program needs to be able to halt its execution either normally (end) or abnormally (abort). If a system call is made to terminate the currently running program abnormally, or if the program runs into a problem and causes an error trap, a dump of memory is sometimes taken and an error message generated. The dump is written to disk and may be examined by a **debugger**—a system program designed to aid the programmer in finding and correcting bugs—to determine the cause of the problem. Under either normal or abnormal circumstances, the operating system must transfer control to the invoking command interpreter. The command interpreter then reads the next command. In an interactive system, the command interpreter simply continues with the next command; it is assumed that the user will issue an appropriate command to respond to any error.

File Management

We first need to be able to create and delete files. Either system call requires the name of the file and perhaps some of the file's attributes. Once the file is created, we need to open it and to use it. We may also read, write, or reposition (rewinding or skipping to the end of the file, for example). Finally, we need to close the file, indicating that we are no longer using it. We may need these same sets of operations for directories if we have a directory structure for organizing files in the file system. In addition, for either files or directories, we need to be able to determine the values of various attributes and perhaps to reset them if necessary. File attributes include the file name, a file type, protection codes, accounting information, and so on.
At least two system calls, get file attribute and set file attribute, are required for this function. Some operating systems provide many more calls, such as calls for file move and copy.

Device Management

A process may need several resources to execute—main memory, disk drives, access to files, and so on. If the resources are available, they can be granted, and control can be returned to the user process. Otherwise, the process will have to wait until sufficient resources are available. The various resources controlled by the operating system can be thought of as devices. Some of these devices are physical devices (for example, tapes), while others can be thought of as abstract or virtual devices (for example, files). If there are multiple users of the system, the system may require us to first request the device, to ensure exclusive use of it. After we are finished with the device, we release it. These functions are similar to the open and system calls for files.

Information Maintenance

Many system calls exist simply for the purpose of transferring information between the user program and the operating system. For example, most systems have a system call to return the current t I m e and date . Other system calls may return information about the system, such as the number of current users, the version number of the operating system, the amount of free memory or disk space, and so on.

In addition, the operating system keeps information about all its processes, and system calls are used to access this information. Generally, calls are also used to reset the process information (get process attributes and set process attributes) .

Communication

There are two common models of inter process communication: the message passing model and the shared-memory model. In the message-passing model, the communicating processes exchange messages with one another to transfer information. Messages can be exchanged between the processes either directly or indirectly through a common mailbox. Before communication can take place, a connection must be opened. The name of the other communicator must be known, be it another process on the same system or a process on another computer connected by a communications network. Each computer in a network has a *host name* by which it is commonly known. A host also has a network identifier, such as an IP address. Similarly, each process has a *process name*, and this name is translated into an identifier by which the operating system can refer to the process. The get host id and get processid system calls do this translation. The identifiers are then passed to the general purpose open and close calls provided by the file system or to specific open connection and close connection system calls, depending on the system's model of communication.

In the shared-memory model, processes use shared memory creates and shared memory attaches system calls to create and gain access to regions of memory owned by other processes. Recall that, normally, the operating system tries to prevent one process from accessing another process's memory. Shared memory requires that two or more processes agree to remove this restriction.
They can then exchange information by reading and writing data in the shared areas. The form of the data and the location are determined by the processes and are not under the operating system's control. The processes are also responsible for ensuring that they are not writing to the same location simultaneously.

1.2.3 System Programs

At the lowest level is hardware. Next are the operating system, then the system programs, and finally the application programs. System programs provide a convenient environment for program development and execution. Some of them are simply user interfaces to system calls; others are considerably more complex. They can be divided into these categories:
• **File management.** These programs create, delete, copy, rename, print, dump, list, and generally manipulate files and directories.

• **Status information.** Some programs simply ask the system for the date, time, amount of available memory or disk space, number of users, or similar status information. Others are more complex, providing detailed performance, logging, and debugging information. Typically, these programs format and print the output to the terminal or other output devices or files or display it in a window of the GUI. Some systems also support a registry, which is used to store and retrieve configuration information.
• **File modification.** Several text editors may be available to create and modify the content of files stored on disk or other storage devices. There may also be special commands to search contents of files or perform transformations of the text.
• **Programming-language support.** Compilers, assemblers, debuggers and interpreters for common programming languages (such as C, C++, Java, Visual Basic, and PERL) are often provided to the user with the operating system.
• **Program loading and execution.** Once a program is assembled or compiled, it must be loaded into memory to be executed. The system may provide absolute loaders, relocatable loaders, linkage editors, and overlay loaders. Debugging systems for either higher-level languages or machine language are needed as well.
• **Communications.** These programs provide the mechanism for creating virtual connections among processes, users, and computer systems. They allow users to send messages to one another's screens, to browse web pages, to send electronic-mail messages, to log in remotely, or to transfer files from one machine to another.

In addition to systems programs, most In addition to systems programs, most operating systems are supplied with programs that are useful in solving common problems or performing common operations. Such programs include web browsers, word processors and text formatters, spreadsheets, database systems, compilers, plotting and statistical-analysis packages, and games. These programs are known as system utilities or application programs.

Q2.a.Explain simple layered, and microkernel structure of the operating system?

1. Simple Structure

- **Overview**: The simple structure is a basic design where there is no clear division of functionality into modules. Instead, all the system components, including process management, memory management, and file systems, are often combined into a single, monolithic system.
- **Characteristics**:
    - **Monolithic Design**: All services and functions are integrated into a single large codebase.
    - **Direct Communication**: There is no strict separation between components, which means they can directly communicate with each other.
    - **Examples**: Early operating systems like MS-DOS and early versions of UNIX used this approach.
- **Advantages**:
    - **Fast Performance**: Due to the lack of abstraction layers, the system can be very fast.
    - **Simple Design**: Easier to develop initially.
- **Disadvantages**:
    - **Complex Maintenance**: As the OS grows, maintaining and debugging becomes difficult due to the lack of modularity.
    - **Poor Security and Reliability**: A single bug can bring down the entire system, as all components are tightly coupled.

2. Layered Structure

- **Overview**: In a layered structure, the OS is divided into layers, each built on top of the lower one. The bottom layer (layer 0) is the hardware, and the topmost layer (layer N) is the user interface.
- **Characteristics**:
    - **Layered Design**: Each layer only interacts with the layer directly below and above it.
    - **Modularity**: Functions and services are separated into different layers, with clear boundaries.
    - **Examples**: The MULTICS system and some versions of UNIX implement a layered approach.
- **Advantages**:
    - **Ease of Maintenance**: Modular design makes it easier to update, debug, and maintain the OS.
    - **Improved Security**: Layers provide a form of isolation; a failure in one layer doesn't necessarily bring down the entire system.
- **Disadvantages**:
    - **Performance Overhead**: Communication between layers adds overhead, potentially slowing down the system.
    - **Rigid Structure**: Changing the order of layers or adding new functionality can be challenging.

3. Microkernel Structure

- **Overview**: The microkernel structure minimizes the OS kernel, keeping only the most essential services in the kernel. All other services, such as device drivers, file systems, and network protocols, run in user space as separate processes.
- **Characteristics**:
  - **Minimal Kernel**: The kernel is small and only includes essential functions like inter-process communication, basic scheduling, and low-level memory management.
  - **User-space Services**: Most OS services run in user space, outside the kernel.
  - **Examples**: Examples include QNX, Minix, and the Mach kernel used in macOS.
- **Advantages**:
  - **Improved Stability**: Since most services run in user space, a failure in one service won't crash the entire system.
  - **Security**: The microkernel provides strong isolation between services, enhancing security.
  - **Flexibility**: It's easier to modify and extend the system since services can be independently updated or replaced.
- **Disadvantages**:
  - **Performance Overhead**: The frequent switching between user space and kernel space (context switching) can lead to performance issues.
  - **Complexity**: The design and development of a microkernel-based OS can be more complex compared to other structures.

Q2.b.What is Virtual Machines? Explain the implementation of Virtual Machines?

A **Virtual Machine (VM)** is a software-based emulation of a physical computer that runs an operating system (OS) and applications just like a physical machine. VMs are created by a software layer known as a **hypervisor** or **virtual machine monitor (VMM)** that abstracts and manages the physical hardware resources, allowing multiple virtual machines to run on a single physical machine simultaneously.

Each VM operates as if it has its own hardware, including CPU, memory, storage, and network interfaces, despite these resources being shared among multiple VMs.

Types of Virtual Machines

1. **System Virtual Machines**: These provide a complete system environment, allowing a full OS to run as if it were on physical hardware. Each VM runs its own OS and can function independently of others.
   - **Examples**: VMware, VirtualBox, Microsoft Hyper-V.
2. **Process Virtual Machines**: These are designed to run a single program or process, providing a platform-independent environment for the application.
   - **Example**: The Java Virtual Machine (JVM), which allows Java applications to run on any device or OS that has a JVM implementation.

Implementation of Virtual Machines

The implementation of virtual machines typically involves the following key components:

1. Hypervisor (Virtual Machine Monitor)

- The hypervisor is the core software that enables virtualization. It abstracts the physical hardware and provides each VM with a virtualized set of hardware resources. There are two main types of hypervisors:
    - **Type 1 (Bare-metal Hypervisors)**: These run directly on the physical hardware and manage VMs directly. Examples include VMware ESXi, Microsoft Hyper-V, and Xen.
    - **Type 2 (Hosted Hypervisors)**: These run on top of a host operating system, which provides hardware abstraction. Examples include VMware Workstation and Oracle VirtualBox.

2. Virtual Hardware

- The hypervisor creates virtual versions of hardware components such as CPUs, memory, storage, and network interfaces. Each VM believes it has exclusive access to these resources, even though they are shared among multiple VMs.
- **CPU Virtualization**: The hypervisor allocates CPU cycles to VMs and can simulate multiple virtual CPUs for each VM.
- **Memory Virtualization**: Each VM is assigned a portion of the physical memory. The hypervisor manages memory allocation and can also use techniques like paging to optimize memory usage.
- **Storage Virtualization**: VMs use virtual disks, which are typically large files on the host machine that simulate a physical hard drive.
- **Network Virtualization**: The hypervisor provides virtual network interfaces to VMs, allowing them to connect to virtual networks and physical networks as needed.

3. Virtual Machine Files

- VMs are typically represented by a set of files stored on the host machine. These files include:
    - **Configuration File**: Defines the VM's settings, such as the amount of memory, number of CPUs, and hardware configurations.
    - **Virtual Disk Files**: Contain the VM's virtual hard disks.
    - **Snapshot Files**: Save the state of a VM at a particular point in time, allowing it to be restored later.

4. Guest Operating System

- Each VM runs its own guest operating system (OS). The guest OS operates as if it were running on physical hardware, unaware that it is running in a virtualized environment.

5. Resource Allocation and Management

- The hypervisor is responsible for managing and allocating physical resources to VMs. It ensures that each VM gets the resources it needs while maintaining isolation between VMs.
- **Overcommitment**: The hypervisor can allocate more virtual resources than the physical hardware has, based on the assumption that not all VMs will fully use their allocated resources simultaneously.

6. Security and Isolation

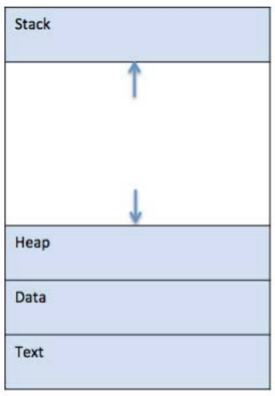
- VMs are isolated from each other, meaning that a failure or security breach in one VM does not affect others. The hypervisor enforces this isolation by managing memory and CPU execution contexts separately for each VM.

Module 2

Q3.a.What is Process? Explain the five state process model with a neat diagram

Process : A process is a program in execution. A process is more than the program code, which is sometimes known as the **text section.** It also includes the current activity, as represented by the value of the **program counter** and the contents of the processor's registers. A process generally also includes the process **stack,** which contains temporary data (such as function parameters, return addresses, and local variables), and a **data section,** which contains global variables. A process may also include a **heap,** which is memory that is dynamically allocated during process run time.

Structure of a process



We emphasize that a program by itself is not a process; a program is a *passive* entity, such as a file containing a list of instructions stored on disk (often called an **executable file),** whereas a process is an *active* entity, with a program counter specifying the next instruction to execute and a set of associated resources. A program becomes a process when an executable file is loaded into memory.

Two common techniques for loading executable files are double-clicking an icon representing the executable file and entering the name of the executable file on the command line (as in prog. exe or a. out.)

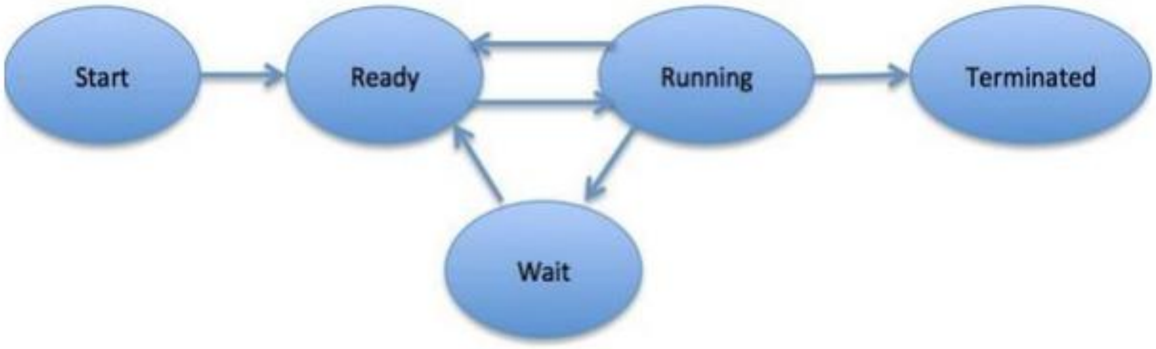
Process State

As a process executes, it changes **state.** The state of a process is defined in part by the current activity of that process. Each process may be in one of the following states:

• **New.** The process is being created.

• **Running.** Instructions are being executed.
• **Waiting.** The process is waiting for some event to occur
(such as an I/O completion or reception of a signal).
• **Ready.** The process is waiting to be assigned to a processor.
• **Terminated.** The process has finished execution.

These names are arbitrary, and they vary across operating systems. The states that they represent are for on all systems, however. Certain operating systems also more finely delineate process states. It is important to realize that only one process can be *running* on any processor at any instant.

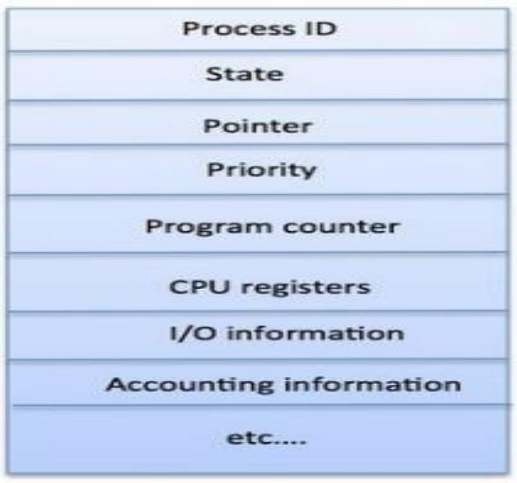


Process State Diagram

Process Control Block

Each process is represented in the operating system by a **process control block (PCB)**—also called a *task control block.*

Process state. The state may be new, ready, running, and waiting, halted, and so on.



Process Control Block (PCB) Diagram

Program counter-The counter indicates the address of the next instruction to be executed for this process.
• CPU **registers-** The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-

purpose registers, plus any condition- code information.

CPU-scheduling information- This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.

Memory-management information- This information may include such information as the value of the base and limit registers, the page tables, or the segment tables, depending on the memory system used by the operating system

Accounting information-This information includes the amount of CPU and real time used, time limits, account members, job or process numbers, and so on.

I/O status information-This information includes the list of I/O devices allocated to the process, a list of open files, and so on.

Q3. b. Consider the following processes, which have arrived at the ready queue with the burst and arrival time given in milliseconds as shown below:

Process	Burst time	Arrival Time
P1	8	0
P2	4	1
P3	9	2
P4	5	3

Draw the Gantt Chart and calculate the average waiting time using the following scheduling algorithms:

i) FCFS ii) SJF Preemptive iii) RR(Q=4)

Solution By Steps

Step 1: FCFS Scheduling

****Gantt Chart:****
```
P1  8
P2  4
P3  9
P4  5
```
Waiting Times:

- P1: 0
- P2: 8
- P3: 12
- P4: 21

Average Waiting Time:
(0 + 8 + 12 + 21)/4 = 41/4 = 10.25

Step 2: SJF (Preemptive) Scheduling
Gantt Chart:
```
P1  1
P2  4
P1  3
P4  5
P3  9
```

Waiting Times:
- P1: 1 + 4 + 5 = 10
- P2: 0
- P3: 1 + 4 + 5 + 9 = 19
- P4: 1 + 4 = 5

Average Waiting Time:
(10 + 0 + 19 + 5)/4 = 34/4 = 8.5

Step 3: RR (Q=4) Scheduling
Gantt Chart:
```
P1  4
P2  4
P1  4
P3  4
P4  4
P3  5
```

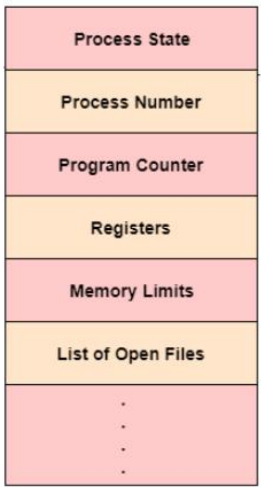
Waiting Times:
- P1: 4
- P2: 0
- P3: 4 + 4 = 8
- P4: 4
Average Waiting Time:
(4 + 0 + 8 + 4)/(4) = 16)/= 4

Final Answer
FCFS Average Waiting Time:** 10.25 ms
SJF (Preemptive) Average Waiting Time:** 8.5 ms
RR (Q=4) Average Waiting Time:** 4 ms

Q4.a.What is Process control Block? Explain the use of PCB in context switching?

Each process is represented in the operating system by a **process control block (PCB)**—also called a *task control block.*

Process state. The state may be new, ready, running, and waiting, halted, and so on.



Process Control Block (PCB)

Program counter-The counter indicates the address of the next instruction to be executed for this process.

• CPU **registers-** The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-purpose registers, plus any condition- code information.

CPU-scheduling information- This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.

Memory-management information- This information may include such information as the value of the base and limit registers, the page tables, or the segment tables, depending on the memory system used by the operating system

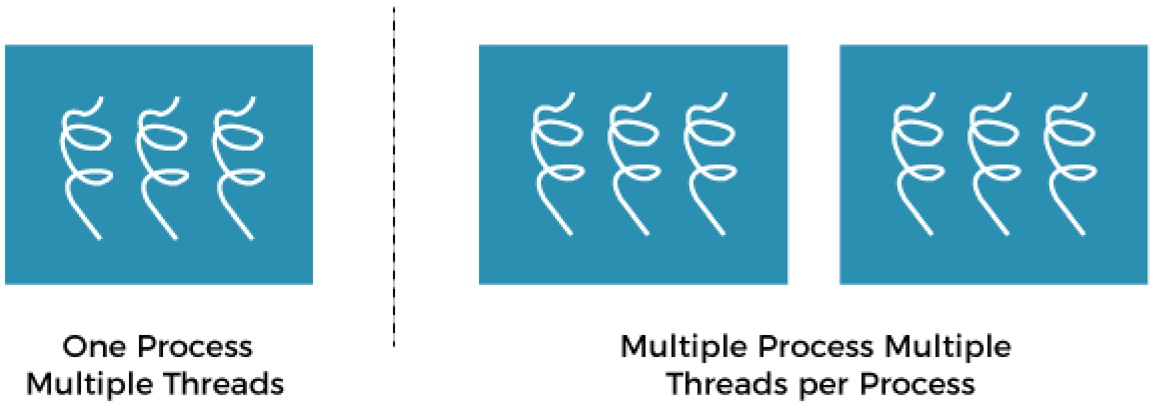
Accounting information-This information includes the amount of CPU and real time used, time limits, account members, job or process numbers, and so on.

I/O status information-This information includes the list of I/O devices allocated to the process, a list of open files, and so on.

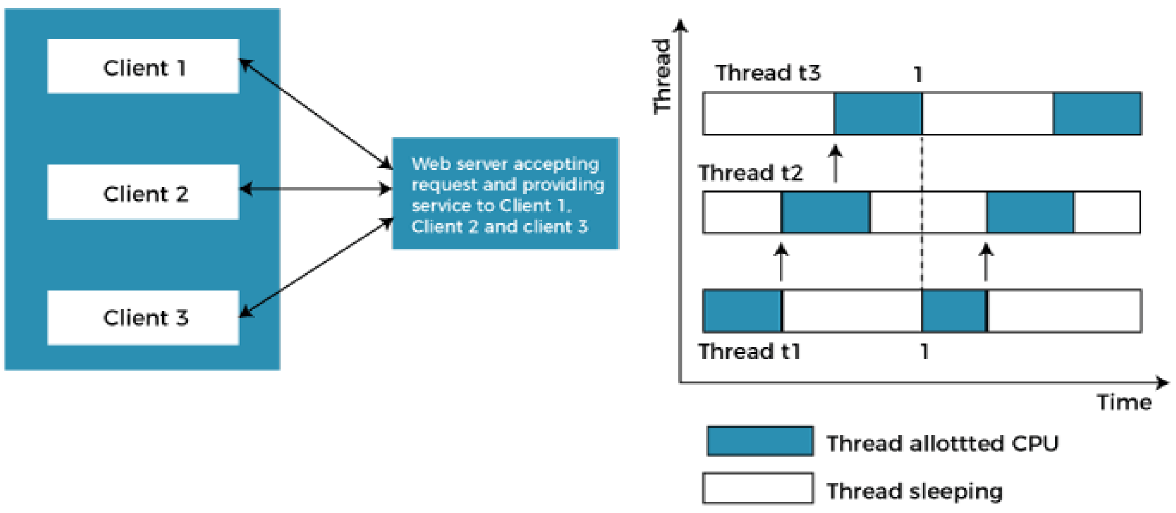
Q4.b.What are user threads and kernel threads? Explain the various multithreading models?

Multithreading Model:

Multithreading allows the application to divide its task into individual threads. In multi-threads, the same process or task can be done by the number of threads, or we can say that there is more than one thread to perform the task in multithreading. With the use of multithreading, multitasking can be achieved.



The main drawback of single threading systems is that only one task can be performed at a time, so to overcome the drawback of this single threading, there is multithreading that allows multiple tasks to be performed.



In the above example, client1, client2, and client3 are accessing the web server without any waiting. In multithreading, several tasks can run at the same time.

In an operating system, threads are divided into the user-level thread and the Kernel-level thread. User-level threads handled independent form above the kernel and thereby managed without any kernel support. On the opposite hand, the operating system directly manages the kernel-level threads. Nevertheless, there must be a form of relationship between user-level and kernel-level threads.

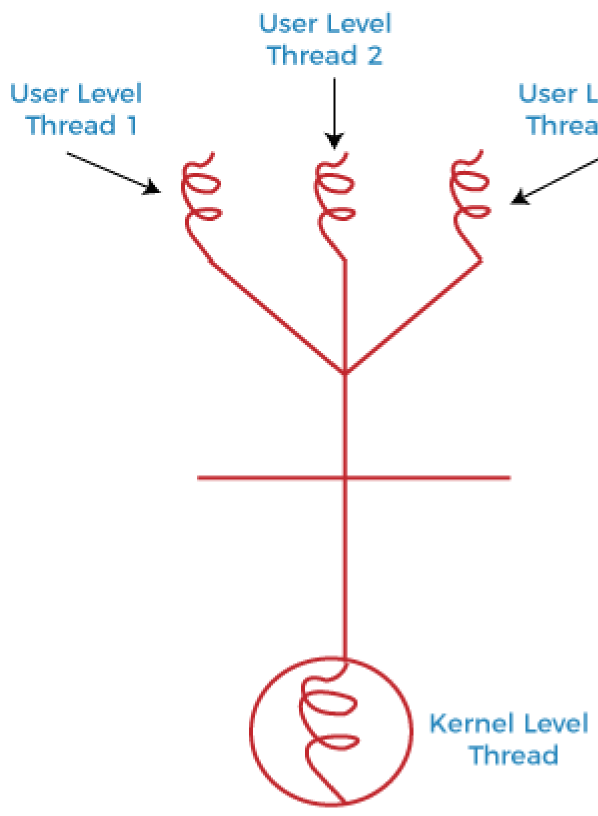
There exists three established multithreading models classifying these relationships are:

- Many to one multithreading model
- One to one multithreading model
- Many to Many multithreading models

Many to one multithreading model:

The many to one model maps many user levels threads to one kernel thread. This type of relationship facilitates an effective context-switching environment, easily implemented even on the simple kernel with no thread support.

The disadvantage of this model is that since there is only one kernel-level thread schedule at any given time, this model cannot take advantage of the hardware acceleration offered by multithreaded processes or multi-processor systems. In this, all the thread management is done in the userspace. If blocking comes, this model blocks the whole system.

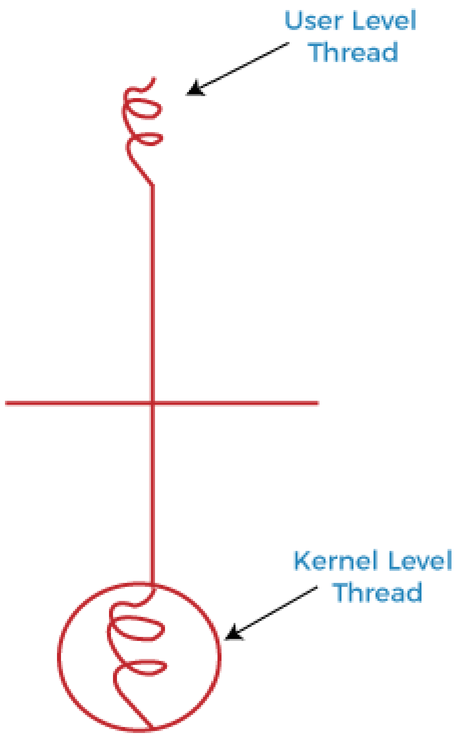


Many-to-one model

In the above figure, the many to one model associates all user-level threads to single kernel-level threads.

One to one multithreading model

The one-to-one model maps a single user-level thread to a single kernel-level thread. This type of relationship facilitates the running of multiple threads in parallel. However, this benefit comes with its drawback. The generation of every new user thread must include creating a corresponding kernel thread causing an overhead, which can hinder the performance of the parent process. Windows series and Linux operating systems try to tackle this problem by limiting the growth of the thread count.

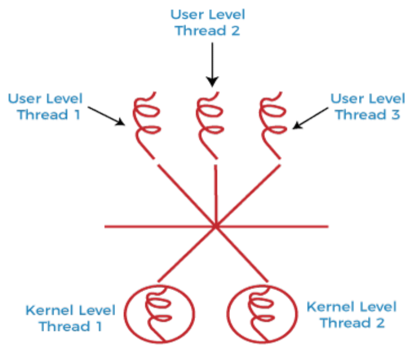


One-to-one model

In the above figure, one model associates that one user-level thread to a single kernel-level thread.

Many to Many Model multithreading model

In this type of model, there are several user-level threads and several kernel-level threads. The number of kernel threads created depends upon a particular application. The developer can create as many threads at both levels but may not be the same. The many to many model is a compromise between the other two models. In this model, if any thread makes a blocking system call, the kernel can schedule another thread for execution. Also, with the introduction of multiple threads, complexity is not present as in the previous models. Though this model allows the creation of multiple kernel threads, true concurrency cannot be achieved by this model. This is because the kernel can schedule only one process at a time.



Many-to-Many model

Many to many versions of the multithreading model associate several user-level threads to the same or much less variety of kernel-level threads in the above figure.

Module 3

Q5.a.What is a critical section problem? Illustrate Peterson's two process solution for a critical section problem?

1. **critical section problem.**
   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 co-operating processes, at a given point of time, only one process must be executing its critical section. If other processes also want to execute its critical section, it must wait until the first one finishes.
   Critical section:
   - It is the part of the program where shared resources are accessed by various processes.
   - It is the place where shred variable, resources are placed.

   Solution to Critical section Problem
   A solution to the critical section problems must satisfy the following three conditions:
   1. Mutual exclusion
   2. Progress
   3. Bounded waiting
   4. No assumption related to H/W speed

   1. Mutual exclusion
   Out of a group of co-operating 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 process wants to execute in critical section than one of these process 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 time is reached, system must grant the process permission to get into its critical section.
   4. No assumption related to H/W speed
   Peterson's Solution
   - Good algorithmic description of solving the problem

   Two process solution
   - Assume that the load and store machine-language instructions are atomic; that is, cannot be interrupted
   - The two processes share two variables:
     int turn;

   Boolean flag[2]
   - The variable turn indicates whose turn it is to enter the critical section
   - The flag array is used to indicate if a process is ready to enter the critical section.
   - flag[i] = true implies that process Pi is ready!

   Algorithm for Process Pi
   do {

```
flag[i] = true;
turn = j;
while (flag[j] && turn = = j);

critical section

flag[i] = false;

remainder section

} while (true);
```

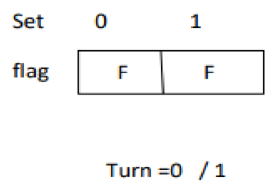
Explanation:

P0

```
while(1)
{
flag[0]=T
turn=1;
while (turn==1 and
flag[1]==T);
critical section
flag[0]=F
}
```

P1

```
while(1)
{
flag[1]=T
turn=0;
while (turn==0 and
flag[0]==T);
critical section
flag[1]=F
}
```



1. Mutual exclusion is preserved
2. The progress requirement is satisfied
3. The bounded –waiting requirement is met.

Q5.b.What are semaphores? Explain the producer-consumer problem and give a solution using semaphores.

Semaphores

- Synchronization tool that provides more sophisticated ways (than Mutex locks)

for process to synchronize their activities.

- Semaphore S – integer variable
- Can only be accessed via two indivisible (atomic) operations
    - wait() and signal()
    - Originally called P() and V()
- Definition of the wait() operation

```
wait(S) {
while (S <= 0)
; // busy wait
S--;
}
```

- Definition of the signal() operation

```
signal(S) {
S++;
}
```

- Counting semaphore – integer value can range over an unrestricted domain

- Binary semaphore – integer value can range only between 0 and 1
    - Same as a mutex lock
- Can solve various synchronization problems
- Consider P1 and P2 that require S1 to happen before S2

  Create a semaphore “synch” initialized to 0

```
P1:
S1;
signal(synch);
P2:
wait(synch);
S2;
```

- Can implement a counting semaphore S as a binary semaphore

Semaphore Implementation

- Must guarantee that no two processes can execute the wait() and signal() on the same semaphore at the same time.
- Thus, the implementation becomes the critical section problem where the wait and signal code are placed in the critical section
    - Could now have busy waiting in critical section implementation
    - But implementation code is short
    - Little busy waiting if critical section rarely occupied
- Note that applications may spend lots of time in critical sections and therefore this is not a good solution

Semaphore Implementation with no Busy waiting

- With each semaphore there is an associated waiting queue
- Each entry in a waiting queue has two data items:
    - value (of type integer)
    - pointer to next record in the list

Two operations:

- block – place the process invoking the operation on the appropriate waiting queue
- wakeup – remove one of processes in the waiting queue and place it in the ready queue

```
 typedef struct{
int value;
struct process *list;
} semaphore;

wait(semaphore *S) {
S->value--;
if (S->value < 0) {
add this process to S->list;
block();
}
}
signal(semaphore *S) {
S->value++;
if (S->value <= 0) {
remove a process P from S->list;
wakeup(P);
```

}
}

Q6.a.What is deadlock? What are the necessary conditions for a deadlock to occur?

- ➢ Every process needs some resources to complete its execution. However, the resource is granted in a sequential order.

1. The process requests for some resource.
2. OS grant the resource if it is available otherwise let the process waits.
3. The process uses it and release on the completion.

- ➢ A Deadlock is a situation where each of the computer process waits for a resource which is being assigned to some another process. In this situation, none of the process gets executed since the resource it needs, is held by some other process which is also waiting for some other resource to be released.
- ➢ Let us assume that there are three processes P1, P2 and P3. There are three different resources R1, R2 and R3. R1 is assigned to P1, R2 is assigned to P2 and R3 is assigned to P3.
- ➢ After some time, P1 demands for R1 which is being used by P2. P1 halts its execution since it can't complete without R2. P2 also demands for R3 which is being used by P3. P2 also stops its execution because it can't continue without R3. P3 also demands for R1 which is being used by P1 therefore P3 also stops its execution.

Necessary conditions for Deadlocks

1. Mutual Exclusion

- ➢ A resource can only be shared in mutually exclusive manner. It implies, if two process cannot use the same resource at the same time.

2. Hold and Wait

- ➢ A process waits for some resources while holding another resource at the same time.

3. No preemption

- ➢ The process which once scheduled will be executed till the completion. No other process can be scheduled by the scheduler meanwhile.

4. Circular Wait

- ➢ All the processes must be waiting for the resources in a cyclic manner so that the last process is waiting for the resource which is being held by the first process. In this scenario, a cycle is being formed among the three processes. None of the process is progressing and they are all waiting. The computer becomes unresponsive since all the processes got blocked.

SYSTEM MODEL

A deadlock occurs when a set of processes is stalled because each process is holding a resource
and waiting for another process to acquire another resource. In the diagram below, for example, Process 1 is holding Resource 1 while Process 2 acquires Resource 2, and Process 2 is waiting for Resource 1.

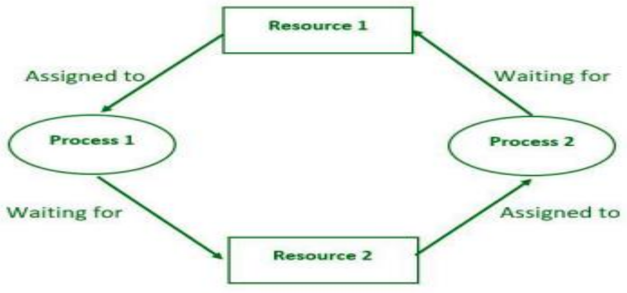


Figure: Deadlock in Operating system

System Model :

- ➢ For the purposes of deadlock discussion, a system can be modeled as a collection of limited resources that can be divided into different categories and allocated to a variety of
processes, each with different requirements.
- ➢ Memory, printers, CPUs, open files, tape drives, CD-ROMs, and other resources are examples of resource categories.
- ➢ By definition, all resources within a category are equivalent, and any of the resources within that category can equally satisfy a request from that category. If this is not the case (i.e. if there is some difference between the resources within a category), then that category must be subdivided further. For example, the term "printers" may need to be subdivided into "laser printers" and "color inkjet printers."
- ➢ Some categories may only have one resource.
- ➢ The kernel keeps track of which resources are free and which are allocated, to which process they are allocated, and a queue of processes waiting for this resource to become available for all kernel-managed resources. Mutexes or wait() and signal() calls can be used to control application-managed resources (i.e. binary or counting semaphores.)

- ➢ When every process in a set is waiting for a resource that is currently assigned to another
process in the set, the set is said to be deadlocked.

Operations:

In normal operation, a process must request a resource before using it and release it when finished, as shown below.

1. Request–
If the request cannot be granted immediately, the process must wait until the resource(s) required to become available. The system, for example, uses the functions open(), malloc(), new(), and request ().

2. Use–
The process makes use of the resource, such as printing to a printer or reading from a file.

3. Release–
The process relinquishes the resource, allowing it to be used by other processes.

Necessary Conditions

There are four conditions that must be met in order to achieve deadlock as follows.

1. Mutual Exclusion –
At least one resource must be kept in a non-shareable state; if another process requests it, it must wait for it to be released.

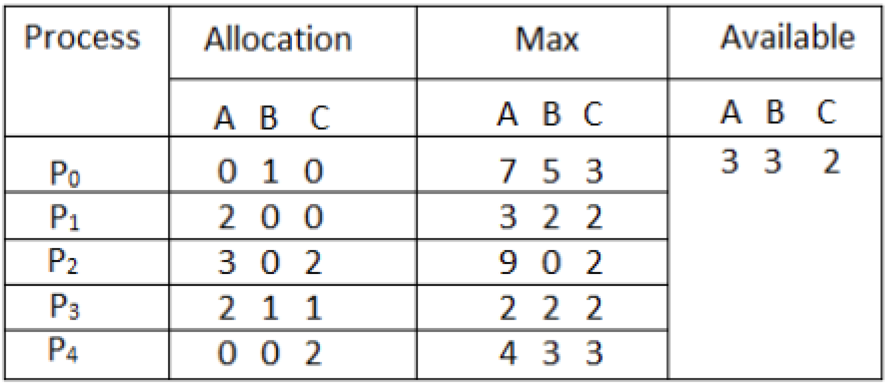
2. Hold and Wait –
A process must hold at least one resource while also waiting for at least one resource that

another process is currently holding.

3. No preemption –
Once a process holds a resource (i.e. after its request is granted), that resource cannot be taken away from that process until the process voluntarily releases it.

4. Circular Wait –
There must be a set of processes P0, P1, P2,..., PN such that every P[I] is waiting for P[(I + 1) percent (N + 1)]. (It is important to note that this condition implies the hold-and-wait condition, but dealing with the four conditions is easier if they are considered separately).

Q6.b.Consider the following snapshot system

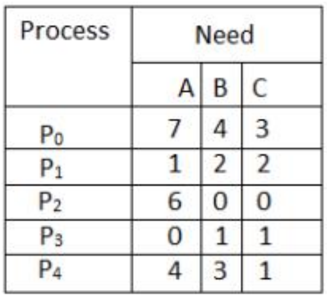


Process	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

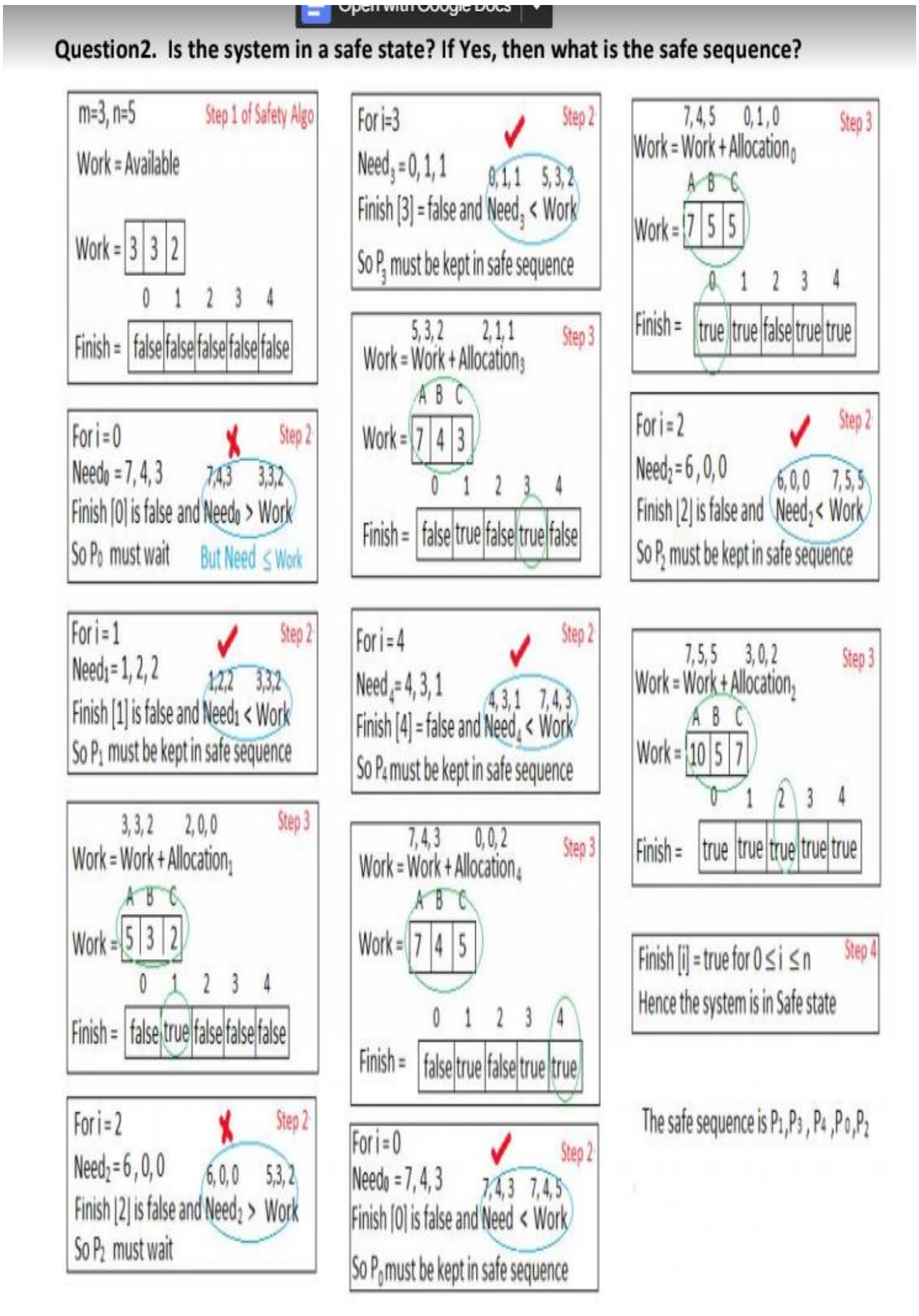
Question1. What will be the content of the Need matrix?

Need [i, j] = Max [i, j] – Allocation [i, j]

So, the content of Need Matrix is:



Process	Need		
	A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1



Question2. Is the system in a safe state? If Yes, then what is the safe sequence?

$m=3, n=5$ — Step 1 of Safety Algo

Work = Available

3	3	2

	0	1	2	3	4
Finish =	false	false	false	false	false

For $i = 0$ ✗ — Step 2

$Need_0 = 7, 4, 3$

7,4,3 3,3,2

Finish [0] is false and $Need_0 > Work$

So P_0 must wait — But Need ≤ Work

For $i = 1$ ✓ — Step 2

$Need_1 = 1, 2, 2$

1,2,2 3,3,2

Finish [1] is false and $Need_1 < Work$

So P_1 must be kept in safe sequence

Step 3

3, 3, 2 2, 0, 0

Work = Work + $Allocation_1$

A	B	C
5	3	2

	0	1	2	3	4
Finish =	false	true	false	false	false

For $i = 2$ ✗ — Step 2

$Need_2 = 6, 0, 0$

6,0,0 5,3,2

Finish [2] is false and $Need_2 > Work$

So P_2 must wait

For $i=3$ ✓ — Step 2

$Need_3 = 0, 1, 1$

0,1,1 5,3,2

Finish [3] = false and $Need_3 < Work$

So P_3 must be kept in safe sequence

Step 3

5, 3, 2 2, 1, 1

Work = Work + $Allocation_3$

A	B	C
7	4	3

	0	1	2	3	4
Finish =	false	true	false	true	false

For $i = 4$ ✓ — Step 2

$Need_4 = 4, 3, 1$

4,3,1 7,4,3

Finish [4] = false and $Need_4 < Work$

So P_4 must be kept in safe sequence

Step 3

7, 4, 3 0, 0, 2

Work = Work + $Allocation_4$

A	B	C
7	4	5

	0	1	2	3	4
Finish =	false	true	false	true	true

For $i = 0$ ✓ — Step 2

$Need_0 = 7, 4, 3$

7,4,3 7,4,5

Finish [0] is false and Need < Work

So P_0 must be kept in safe sequence

Step 3

7, 4, 5 0, 1, 0

Work = Work + $Allocation_0$

A	B	C
7	5	5

	0	1	2	3	4
Finish =	true	true	false	true	true

For $i = 2$ ✓ — Step 2

$Need_2 = 6, 0, 0$

6,0,0 7,5,5

Finish [2] is false and $Need_2 < Work$

So P_2 must be kept in safe sequence

Step 3

7, 5, 5 3, 0, 2

Work = Work + $Allocation_2$

A	B	C
10	5	7

	0	1	2	3	4
Finish =	true	true	true	true	true

Step 4

Finish [i] = true for $0 \le i \le n$

Hence the system is in Safe state

The safe sequence is P_1, P_3, P_4, P_0, P_2

Module 4

Q7.a.Write a C Program to simulate the multiprogramming with variable member of tasks(MVT) memory management technique. Given the size of the memory size of OS, number of processes and size of each process, calculate the external fragmentation

```
#include <stdio.h>

#define MAX 100

int main() {
   int totalMemory, osMemory, numberOfProcesses;
   int availableMemory, processMemory[MAX], i;
   int externalFragmentation = 0;

   // Input total memory size and memory reserved for OS
   printf("Enter the total memory size: ");
   scanf("%d", &totalMemory);
   printf("Enter the memory reserved for OS: ");
   scanf("%d", &osMemory);

   // Calculate available memory after OS allocation
   availableMemory = totalMemory - osMemory;

   // Input number of processes
   printf("Enter the number of processes: ");
   scanf("%d", &numberOfProcesses);

   // Input the size of each process
   for (i = 0; i < numberOfProcesses; i++) {
      printf("Enter the memory required for process %d: ", i + 1);
      scanf("%d", &processMemory[i]);
   }

   // Allocate memory to processes
   for (i = 0; i < numberOfProcesses; i++) {
      if (processMemory[i] <= availableMemory) {
         printf("Process %d is allocated %d units of memory.\n", i + 1, processMemory[i]);
         availableMemory -= processMemory[i];
      } else {
         printf("Process %d cannot be allocated memory due to insufficient available
memory.\n", i + 1);
         externalFragmentation += processMemory[i];
      }
   }

   // Display results
   printf("\nTotal memory: %d\n", totalMemory);
   printf("Memory reserved for OS: %d\n", osMemory);
   printf("Remaining available memory: %d\n", availableMemory);
```

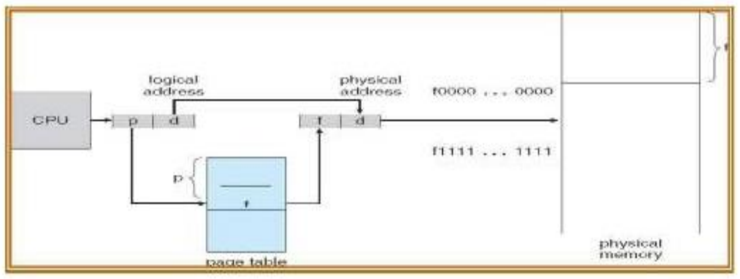
```
    printf("External Fragmentation: %d\n", externalFragmentation);

    return 0;
}
```

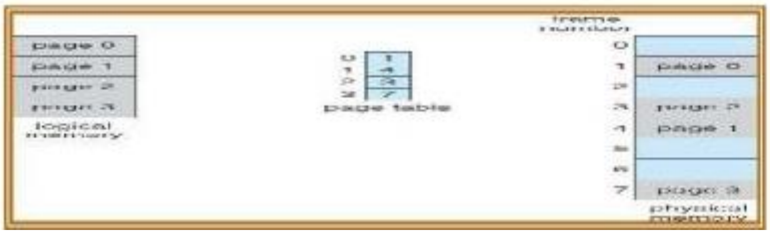
Q7.b.What is paging? Explain the paging hardware with a neat diagram?

- Paging is a memory management scheme that permits the physical address space of a process to be non-contiguous. Support for paging is handled by hardware.
- It is used to avoid external fragmentation.
- Paging avoids the considerable problem of fitting the varying sized memory chunks on to the backing store.
- When some code or date residing in main memory need to be swapped out, space must be found on backing store.

  Basic Method:
  - Physical memory is broken in to fixed sized blocks called frames (f).
  - Logical memory is broken in to blocks of same size called pages (p).
  - When a process is to be executed its pages are loaded in to available frames from backing store.
  - The blocking store is also divided in to fixed-sized blocks of same size as memory frames.
  - The following figure shows paging hardware:



- Logical address generated by the CPU is divided in to two parts: page number (p) and page offset (d).
- The page number (p) is used as index to the page table. The page table contains base address of each page in physical memory. This base address is combined with the page offset to define the physical memory i.e., sent to the memory unit.
- The page size is defined by the hardware. The size of a power of 2, varying between 512 bytes and 10Mb per page.
- If the size of logical address space is 2^m address unit and page size is 2^n, then high order m-n designates the page number and n low order bits represents page offset



Eg:-To show how to map logical memory in to physical memory consider a page size of 4 bytes and physical memory of 32 bytes (8 pages).

a. Logical address 0 is page 0 and offset 0. Page 0 is in frame 5. The logical address 0 maps to physical address 20. [(5*4) + 0].
b. Logical address 3 is page 0 and offset 3 maps to physical address 23 [(5*4) + 3].

Hardwar e Support for Pag i ng:

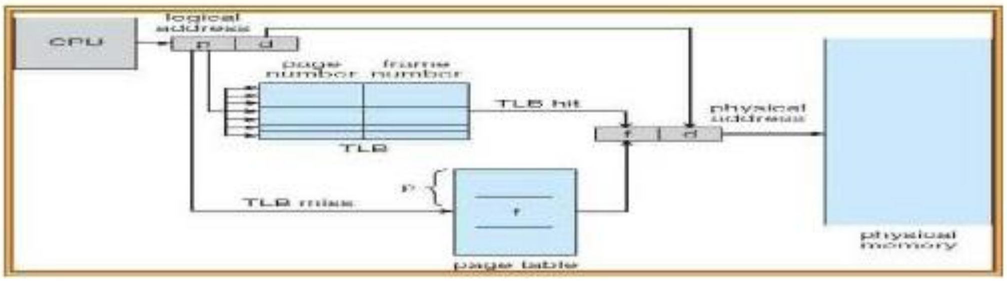
The hardware implementation of the page table can be done in several ways:

1. The simplest method is that the page table is implemented as a set of dedicated registers. These registers must be built with very high speed logic for making paging address translation. Every accessed memory must go through paging map. The use of registers for page table is satisfactory if the page table is small.

2. If the page table is large then the use of registers is not visible. So the page table is kept in the main memory and a page table base register [PTBR] points to the page table. Changing the page table requires only one register which reduces the context switching type. The problem with this approach is the time required to access memory location. To access a location [i] first we have to index the page table using PTBR offset. It gives the frame number which is combined with the page offset to produce the actual address. Thus we need two memory accesses for a byte.

3. The only solution is to use special, fast, lookup hardware cache called translation look a side buffer[TLB]or associative register. TLB is built with associative register with high speed memory. Each register contains two paths a ke y and a value.

and a value.



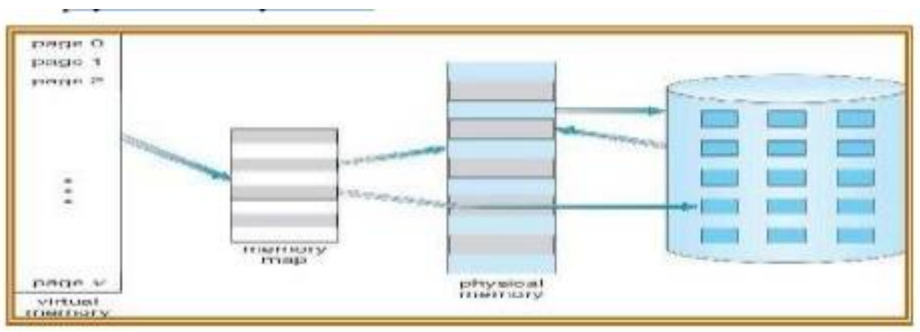
When an associative register is presented with an item, it is compared with all the key values, if found the corresponding value field is return and searching is fast. TLB is used with the page table as follows:

- TLB contains only few page table entries.
- When a logical address is generated by the CPU, its page number along with the frame number is added to TLB. If the page number is found its frame memory is used to access the actual memory.
- If the page number is not in the TLB (TLB miss) the memory reference to the page table is made. When the frame number is obtained use can use it to access the memory.
- If the TLB is full of entries the OS must select anyone for replacement.

- ➢ Each time a new page table is selected the TLB must be flushed [erased] to ensure that next executing process do not use wrong information.
- ➢ The percentage of time that a page number is found in the TLB is called HIT ratio.

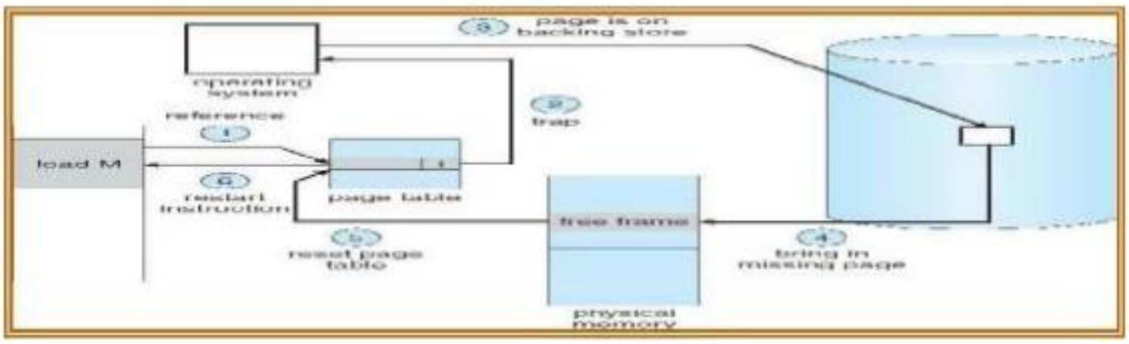
Q8.a.What is demand paging? Explain how demand paging can be implemented

A demand paging is similar to paging system with swapping when we want to execute a process we swap the process the in to memory otherwise it will not be loaded in to memory.
A swapper manipulates the entire processes, where as a pager manipulates individual pages of the process.
Basic concept: Instead of swapping the whole process the pager swaps only the necessary pages in to memory. Thus it avoids reading unused pages and decreases the swap time and amount of physical memory needed.



The valid-invalid bit scheme can be used to distinguish between the pages that are on the disk and that are in memory.
• If the bit is valid then the page is both legal and is in memory.
•If the bit is invalid then either page is not valid or is valid but is currently on the disk.
Marking a page as invalid will have no effect if the processes never access to that page. Suppose if it access the page which is marked invalid, causes a page fault trap. This may result in failure of OS to bring the desired page in to memory.

The step for handling page fault is straight forward and is given below:
1. We check the internal table of the process to determine whether the reference made is valid or invalid.
2. If invalid terminate the process,. If valid, then the page is not yet loaded and we now page it in.
3. We find a free frame.
4. We schedule disk operation to read the desired page in to newly allocated frame.
5. When disk reed is complete, we modify the internal table kept with the process to indicate that the page is now in memory.
6. We restart the instruction which was interrupted by illegal address trap. The process can now access the page. In extreme cases, we start the process without pages in memory. When the OS points to the instruction of process it generates a page fault. After this page is brought in to memory the process continues to execute, faulting as necessary until every demand paging i.e., it never brings the page in to memory until it is required.

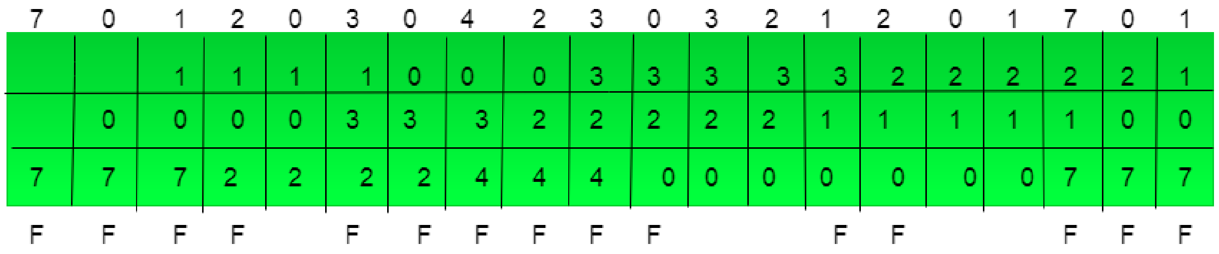


Hardware support: For demand paging the same hardware is required as paging and swapping. 1. Page table:-Has the ability to mark an entry invalid through valid-invalid bit.
2. Secondary memory:-This holds the pages that are not present in main memory.
Performance of demand paging:
Demand paging can have significant effect on the performance of the computer system. Let P be the probability of the page fault (0<=P<=1) Effective access time = (1-P) * ma + P * page fault. Where P = page fault and ma = memory access time. Effective access time is directly proportional to page fault rate. It is important to keep page fault rate low in demand paging

A page fault causes the following sequence to occur:

- Trap to the OS. Save the user registers and process state.
- Determine that the interrupt was a page fault.
- Checks the page references were legal and determine the location of page on disk. Issue a read from disk to a free frame.
- If waiting, allocate the CPU to some other user.
- Interrupt from the disk. Save the registers and process states of other users.
- Determine that the interrupt was from the disk.
- Correct the page table and other table to show that the desired page is now in memory.
- Wait for the CPU to be allocated to this process again.
- Restore the user register process state and new page table then resume the interrupted instruction.

Q8.b. Consider the following reference string:
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
How many page faults would occur in case of i)FIFO ii)Optimal algorithm?



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

In FIFO Algorithm 15 page fault will occur

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

In Optimal Algorithm 9 page fault will occur

LRU Algorithm:

The **Least Recently Used (LRU)** page replacement algorithm replaces the page that hasn't been used for the longest time.

Assumptions:

Let's assume the number of page frames (slots in memory) is **3**, which is common in such problems.

Step-by-Step Execution:

We'll keep track of the pages in memory (with 3 frames) and count the page faults.

Reference	**Page Frames**	**Page Fault?**
7	7 - -	Yes
0	7 0 -	Yes
1	7 0 1	Yes
2	2 0 1	Yes
0	2 0 1	No
3	2 3 1	Yes
0	2 3 0	Yes
4	4 3 0	Yes
2	4 2 0	Yes
3	4 2 3	Yes
0	0 2 3	Yes
3	0 2 3	No
2	0 2 3	No
1	0 1 3	Yes
2	0 1 2	Yes
0	0 1 2	No
1	0 1 2	No
7	7 1 2	Yes
0	7 0 2	Yes
1	7 0 1	Yes

Summary:

- **Total Page Faults**: 15

So, the LRU page replacement algorithm would result in **15 page faults** for the given reference string with 3 page frames.

Module 5

Q9.a. What are the various access methods used to access files?

Access Methods

The file information is accessed and read into computer memory. The information in the file can be accessed in several ways.

a) Sequential Access

- Here information in the file is processed in order, one record after the other.
- This mode of access is a common method; for example, editors and compilers usually access files in this fashion.
- A sequential access file emulates magnetic tape operation, and generally supports a few operations: o read next - read a record and advance the file pointer to the next position. o write next - write a record to the end of file and advance the file pointer to the next position. o skip n records - May or may not be supported. 'n' may be limited to positive numbers, or may be limited to +/- 1.

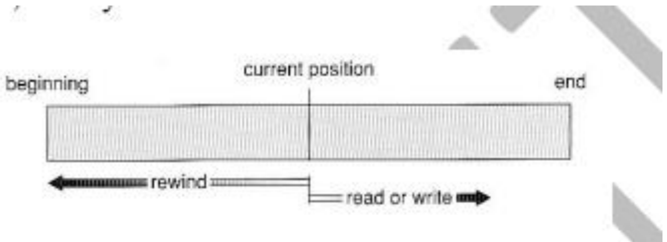


Figure 10.3 Sequential-access file.

b) Direct Access

A file is made up of fixed-length logical records that allow programs to read and write records randomly. The records can be rapidly accessed in any order.

Direct access are of great use for immediate access to large amount of information. Eg : Database file. When a query occurs, the query is computed and only the selected rows are access directly to provide the desired information. Operations supported include:

- read n - read record number n. (position the cursor to n and then read the record)
- write n - write record number n. (position the cursor to n and then write the record)
- jump to record n – move to nth record (n- could be 0 or the end of file)
- If the record length is L, there is a request for record 'N'. Then the direct access to the starting byte of record 'N' is at L*(N-1) Eg: if 3rd record is required and length of each record(L) is 50, then the starting position of 3rd record is L*(N-1) Address = 50*(3-1) = 100.

c) Other Access Methods (Indexed method)

- These methods generally involve the construction of an index for the file called index file.
- The index file is like an index page of a book, which contains key and address. To find a record in the file, we first search the index and then use the pointer to access the record directly and find the desired record.

- An indexed access scheme can be easily built on top of a direct access system.
- For very large files, the index file itself is very large. The solution to this is to create an index for index file. i.e. multi-level indexing.

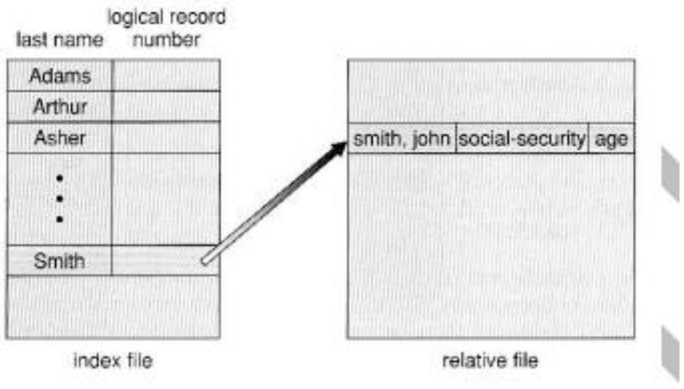


Figure 10.5 Example of index and relative files.

Q9.b. Explain the various directory structures with neat diagrams.

Directory Structures –

a) Single-Level Directory

• It is the simplest directory structure.
• All files are contained in the same directory, which is easy to support and understand. The limitations of this structure is that –
• All files are in the same directory must have unique names.
• 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.

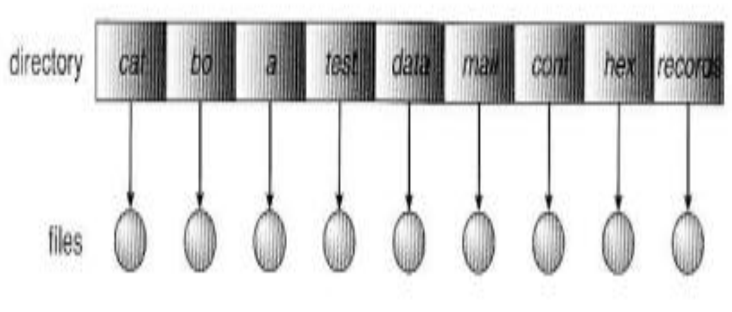


Figure 10.7 Single-level directory.

b) Two-Level Directory

• Each user gets their own directory space - user file directory(UFD)
• File names only need to be unique within a given user's directory.
• A master file directory(MFD) is used to keep track of each users directory, and must be maintained when users are added to or removed from the system.
• When a user refers to a particular file, only his own UFD is searched.
• All the files 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.
- This structure 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

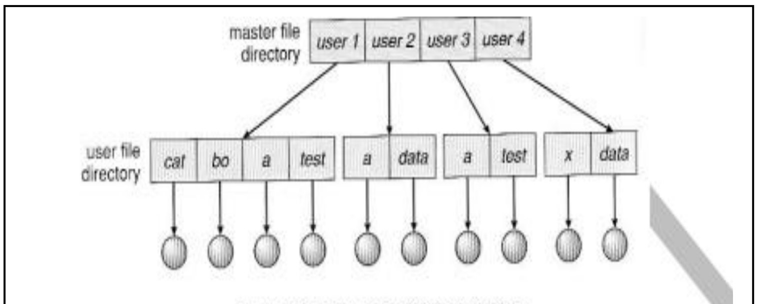


Figure 10.8 Two-level directory structure.

.

c) Tree-Structured Directories

• A tree structure 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.
• 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.
• For example, in the tree-structured file system of figure below if the current directory is root/spell/mail, then the relative path name is prt/first and the files absolute path name root/spell/mail/prt/jirst.

- Directories are stored the same as any other file in the system, except there is a bit that identifies them as directories, and they have some special structure that the OS understands.

• One question for consideration is whether or not to allow the removal of directories that are not empty - Windows requires that directories be emptied first, and UNIX provides an option for deleting entire sub-trees.

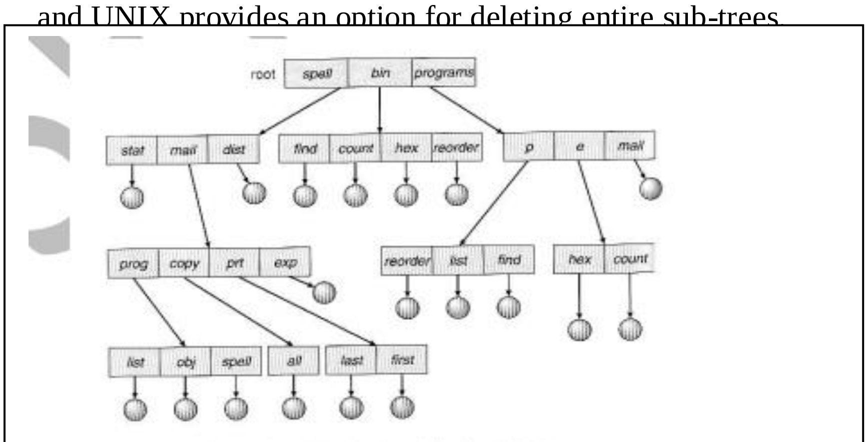


Figure 10.9 Tree-structured directory structure.

d) Acyclic-Graph Directories

• When the same files need to be accessed in more than one place in the directory structure (e.g. because they are being shared by more than one user), it can be useful to provide an acyclic-graph structure. (Note the directed arcs from parent to child.)

- UNIX provides two types of links (pointer to another file)for implementing the acyclic-graph structure.
- A hard link (usually just called a link) involves multiple directory entries that both refer to the same file. Hard links are only valid for ordinary files in the same filesystem.
- A symbolic link, that involves a special file, containing information about where to find the linked file. Symbolic links may be used to link directories and/or files in other filesystems, as well as ordinary files in the current filesystem.
- Windows only supports symbolic links, termed shortcuts.
- Hard links require a reference count, or link count for each file, keeping track of how many directory entries are currently referring to this file. Whenever one of the references is removed the link count is reduced, and when it reaches zero, the disk space can be reclaimed.

For symbolic links there is some question as to what to do with the symbolic links when the original file is moved or deleted: o One option is to find all the symbolic links and adjust them also.
Another is to leave the symbolic links dangling, and discover that they are no longer valid the next time they are used. o What if the original file is removed, and replaced with another file having the same name before the symbolic link is next used? Another approach to deletion is to preserve the file until all references to it are deleted. To implement this approach, we must have some mechanism for determining that the last reference to the file has been deleted. When a link or a copy of the directory entry is established, a new entry is added to the filereference list. When a link or directory entry is deleted, we remove its entry on the list. The file is deleted when its file-reference list is empty.

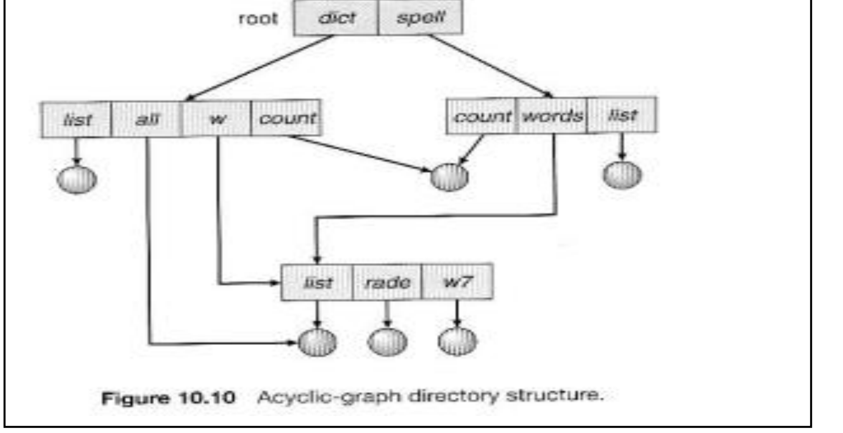


Figure 10.10 Acyclic-graph directory structure.

e) General Graph Directory

• If cycles are allowed in the graphs, then several problems can arise: o Search algorithms can go into infinite loops.
One solution is to not follow links in search algorithms. (Or not to follow symbolic links, and to only allow symbolic links to refer to directories)

- Sub-trees can become disconnected from the rest of the tree and still not have their reference counts reduced to zero. Periodic garbage collection is required to detect and resolve this problem. (chkdsk in DOS and fsck in UNIX search for these problems, among others, even though cycles are not supposed to be allowed in either system. Disconnected disk blocks that are not marked as free are added back to the file systems with made-up file names, and can usually be safely deleted.)

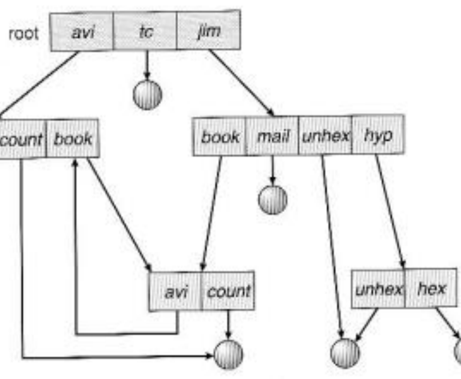


Figure 10.11 General graph directory.

Q10.a. Explain the file allocation methods with neat diagrams.

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:

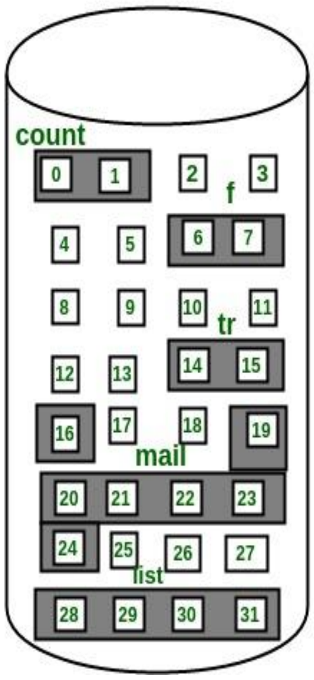
1. Contiguous Allocation

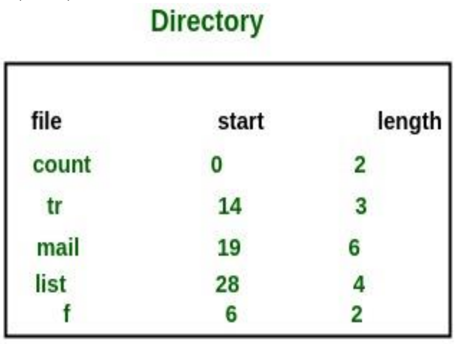
In this scheme, each file occupies a contiguous set of blocks on the disk. For example, if a file requires n blocks and is given a block b as the starting location, then the blocks assigned to the file will be: b, b+1, b+2,……b+n-1. This means that given the starting block address and the length of the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

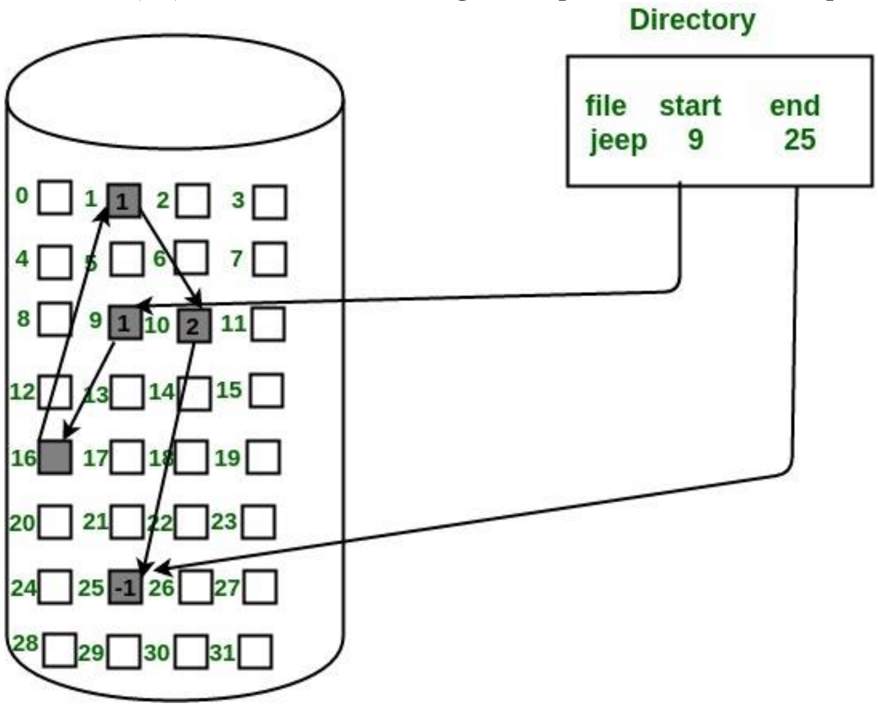
Disadvantages:

- 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.

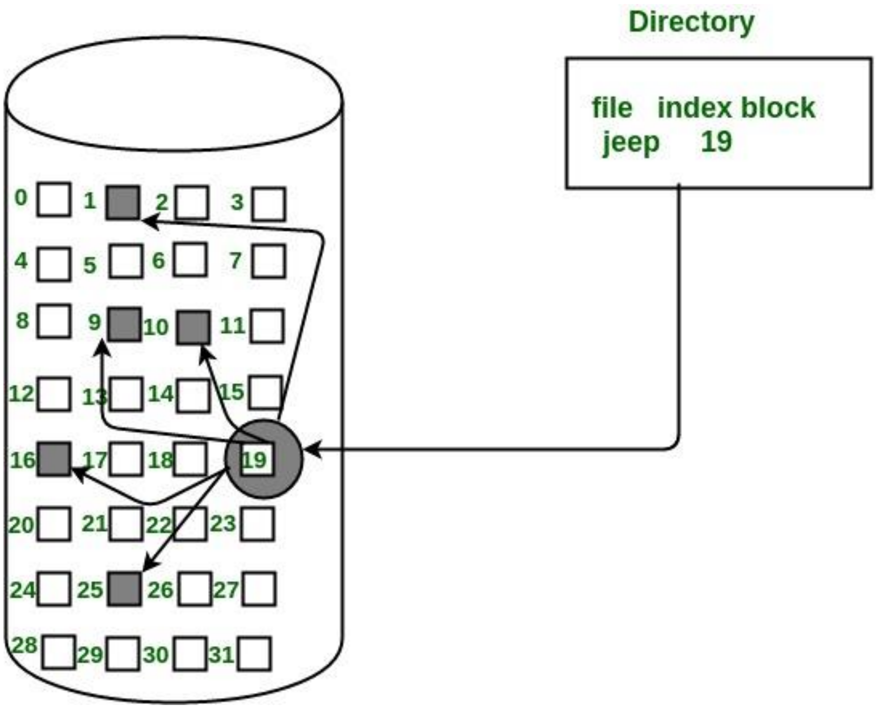
- The file ‘jeep’ in following image shows how the blocks are randomly distributed. The last block (25) contains -1 indicating a null pointer and does not point to any other block.



- 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.

Q10.b. Explain how free space management is done using i) Bit Vector ii) Linked List iii) Grouping and iv) Counting.

Free space management is a critical aspect of operating systems as it involves managing the available storage space on the hard disk or other secondary storage devices. The operating system uses various techniques to manage free space and optimize the use of storage devices.

The system keeps tracks of the free disk blocks for allocating space to files when they are created. Also, to reuse the space released from deleting the files, free space management becomes crucial. The system maintains a free space list which keeps track of the disk blocks that are not allocated to some file or directory. The free space list can be implemented mainly as:

1. **Bitmap or Bit vector** – A Bitmap or Bit Vector is series or collection of bits where each bit corresponds to a disk block. The bit can take two values: 0 and 1: 0 indicates that the block is allocated and 1 indicates a free block. The given instance of disk blocks on the disk in Figure 1 (where green blocks are allocated) can be represented by a bitmap of 16 bits as: 0000111000000110.

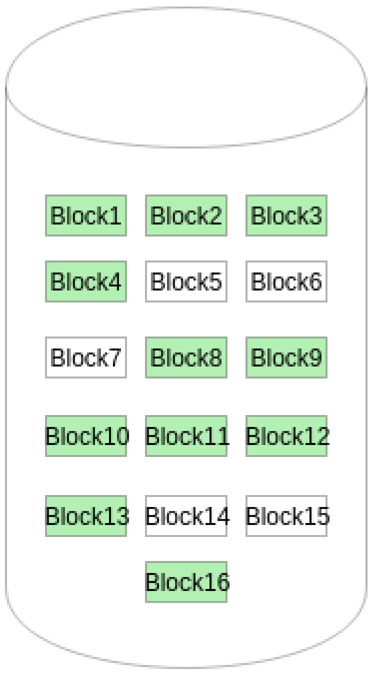


Figure - 1

Advantages –

- Simple to understand.
- Finding the first free block is efficient. It requires scanning the words (a group of 8 bits) in a bitmap for a non-zero word. (A 0-valued word has all bits 0). The first free block is then found by scanning for the first 1 bit in the non-zero word.

2. **Linked List** – In this approach, the free disk blocks are linked together i.e. a free block contains a pointer to the next free block. The block number of the very first disk block is stored at a separate location on disk and is also coached in memory.

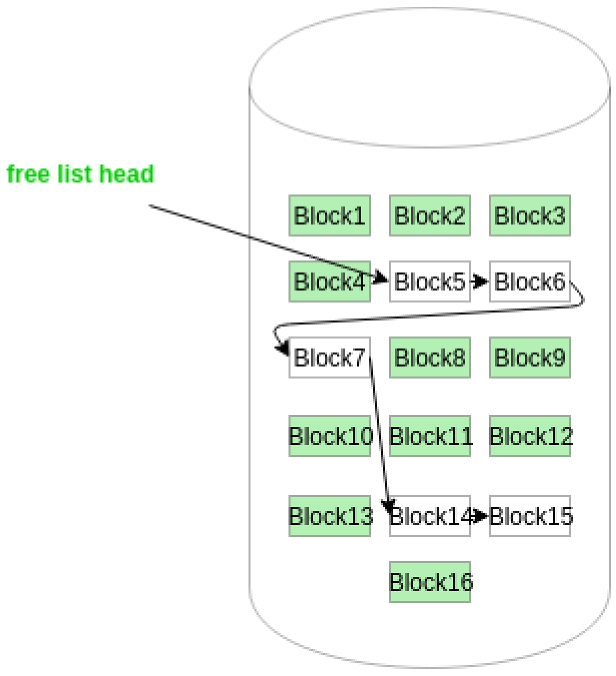


Figure - 2

In Figure-2, the free space list head points to Block 5 which points to Block 6, the next free block and so on. The last free block would contain a null pointer indicating the end of free list. A drawback of this method is the I/O required for free space list traversal.

3. **Grouping –** This approach stores the address of the free blocks in the first free block. The first free block stores the address of some, say n free blocks. Out of these n blocks, the first n-1 blocks are actually free and the last block contains the address of next free n blocks. An advantage of this approach is that the addresses of a group of free disk blocks can be found easily.
4. **Counting –** This approach stores the address of the first free disk block and a number n of free contiguous disk blocks that follow the first block. Every entry in the list would contain:

   Address of first free disk block

   A number n

   Here are some advantages and disadvantages of free space management techniques in operating systems:

   Advantages:

   - Efficient use of storage space: Free space management techniques help to optimize the use of storage space on the hard disk or other secondary storage devices.
   - Easy to implement: Some techniques, such as linked allocation, are simple to implement and require less overhead in terms of processing and memory resources.
   - Faster access to files: Techniques such as contiguous allocation can help to reduce disk fragmentation and improve access time to files.

   Disadvantages:

   - Fragmentation: Techniques such as linked allocation can lead to fragmentation of disk space, which can decrease the efficiency of storage devices.
   - Overhead: Some techniques, such as indexed allocation, require additional overhead in terms of memory and processing resources to maintain index blocks.
   - Limited scalability: Some techniques, such as FAT, have limited scalability in terms of the number of files that can be stored on the disk.
   - Risk of data loss: In some cases, such as with contiguous allocation, if a file becomes corrupted or damaged, it may be difficult to recover the data.
   - Overall, the choice of free space management technique depends on the specific requirements of the operating system and the storage devices being used. While some techniques may offer advantages in terms of efficiency and speed, they may also have limitations and drawbacks that need to be considered.