CBCS SCHEME

USN | | | | | | | | | | |

BCS303

Third Semester B.E./B.Tech. Degree Examination, Dec.2024/Jan.2025

Operating Systems

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.

		Module – 1	M	L	C
Q.1	a.	Define Operating System. Explain dual mode of operating systems with a neat diagram.	06	L1 L2	CO1
	b.	Distinguish between the following terms: i) Multiprogramming and Multitasking ii) Multiprocessor and Clustered system	06	L2	CO1
	c.	Explain with a neat diagram VM-WARE Architecture.	08	L1 L2	CO1
		OR			
Q.2	a.	List and explain the services provided by OS for the user and efficient operation of system.	06	L2	CO1
	b.	Explain the different computing equipments.	06	L2	CO1
	c.	What are systems calls? List and explain the different types of systems calls.	08	L1 L2	CO1
		Module – 2			
Q.3	a.	What is process? Explain process state diagram and process control block with a neat diagram.	10	L1 L2	CO2
	b.	What is interprocess communication? Explain direct and indirect communication with respect to message passing system.	10	L1 L2	CO2
		OR			
Q.4	a.	List and explain the different types of multithreading models.	06	L1 L2	CO2
	b.	Calculate the average waiting time and average turnaround time by drawing the Gantt-chart using FCFS, SJF, RR (Q = 4ms) and priority scheduling (Higher Number is having highest priority).	14	L3	CO2

Process	B.T. (ms)	Priority
P_1	24	1
P_2	03	2
P_3	03	3

		Module – 3	M	L	C
Q.5	a.	What is critical section? Give the Peterson's solution to 2 processes critical section problem.	05	L1 L2	CO3
	b.	Explain Reader's and Writer's problem in detail.	07	L2	CO3
	c.	What is semaphore? Discuss the solution to the classical dinning philosopher problem.	08	L1 L2	CO3

		OR			
Q.6	**a.**	What is a Deadlock? What are the necessary conditions for the deadlock to occur?	**06**	**L1 L2**	**CO3**
	b.	Consider the following snap shot of the system. Process: Allocation (A B C) / Max (A B C) / Available (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: i) What is the content of the matrix need? ii) Is the system on a safe state? If so, find safe sequence. iii) If P_1 requirements for (1, 0, 2) additional resources can P_1 be granted.	**14**	**L3**	**CO2**
		Module – 4			
Q.7	**a.**	What is paging? Explain with a neat diagram paging hardware with TLB.	**10**	**L1 L2**	**CO4**
	b.	Explain the different strategies used to select a free hole from available holes.	**05**	**L1**	**CO4**
	c.	What is Fragmentation? List and explain its types.	**05**	**L2**	**CO4**
		OR			
Q.8	**a.**	What is page fault? With a neat diagram explain steps in handling page fault.	**08**	**L2**	**CO4**
	b.	Consider the page reference string for a memory with 3 frames determine the number of page faults using FIFO, optimal and LRU replacement algorithms. Which algorithms is more efficient? 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1	**12**	**L3**	**CO4**
		Module – 5			
Q.9	**a.**	Define File. List and explain different file operations and file attributes.	**10**	**L1**	**CO5**
	b.	Explain the different file allocation methods.	**10**	**L2**	**CO5**
		OR			
Q.10	**a.**	What is Access Matrix? Explain the implementation of Access Matrix.	**10**	**L2**	**CO5**
	b.	A drive has 5000 cylinders numbered 0 to 4999. The drive is currently servicing at a request 143 and previously served a request at 125. The queue of pending request in FIFO order. 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130 starting from current head position. What is the total distance travelled (in cylinders) by a disk arm to satisfy the request using FCFS, SSTF, SCAN, LOOK and C-Look algorithm	**10**	**L3**	**CO5**

* * * * *

Third Semester B.E. Degree Examination
Operating Systems (BCS303)

TIME: 03 Hours **Max. Marks: 100**

Note:

01. Answer any FIVE full questions, choosing at least ONE question from each MODULE.

MODULE-1

1a .Define operating Systems. Explain the dual-mode operating system with a neat diagram. **6 Marks**

Answer:

A program that acts as an intermediary between a user of a computer and the computer hardware. An operating System is a collection of system programs that together control the operations of a computer systems.

Some examples of operating systems are UNIX, Mach, MS-DOS, MS-Windows, Windows/NT, Chicago, OS/2, MacOS, VMS, MVS, and VM.

I The **Dual-Mode** taken by most computer systems is to provide hardware support that allows us to differentiate among various modes of execution.

At the very least, we need two separate modes of operation: **user mode and kernel mode** (also called supervisor mode, system mode, or privileged mode).

A bit, called the mode bit is added to the hardware of the computer to indicate the current mode: kernel (0) or user (1). With the mode bit, we are able to distinguish between a task that is executed on behalf of the operating system and one that is executed on behalf of the user.

When the computer system is executing on behalf of a user application, the system is in user mode. However, when a user application requests a service from the operating system (via a system call), it must transition from user to kernel mode to fulfill the request.

This is shown in Figure below. As we shall see, this architectural enhancement is useful for many other aspects of system operation as well.

At system boot time, the hardware starts in kernel mode. The operating system is then loaded and starts user applications in user mode. Whenever a trap or interrupt occurs, the hardware switches from user mode to kernel mode (that is, changes the state of the mode bit to 0). Thus, whenever the operating system gains control of the computer, it is in kernel mode. The system always switches to user mode (by setting the mode bit to 1) before passing control to a user program.

The dual mode of operation provides us with the means for protecting the operating system from errant users-and errant users from one another.

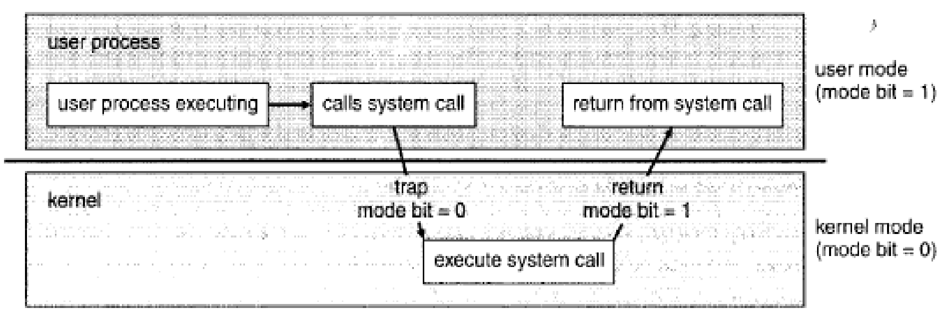


Figure: Transition from user to kernel mode.

1. (b). Distinguish between the following terms.
(i) Multiprogramming and Multitasking
(ii) Multiprocessor System and Clustered System. **6M**

Answer:

i. **Multitasking System and Multiprogramming**: Time sharing (or multitasking) is a logical extension of multiprogramming. In time-sharing systems, the CPU executes multiple jobs by switching among them, but the switches occur so frequently that the users can interact with each program while it is running. A time-shared operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time-shared computer. Each user has at least one separate program in memory. A program loaded into memory and executing is called a process.

Time-sharing and multiprogramming require several jobs to be kept simultaneously in memory. Since in general main memory is too small to accommodate all jobs, the jobs are kept initially on the disk in the job pool.

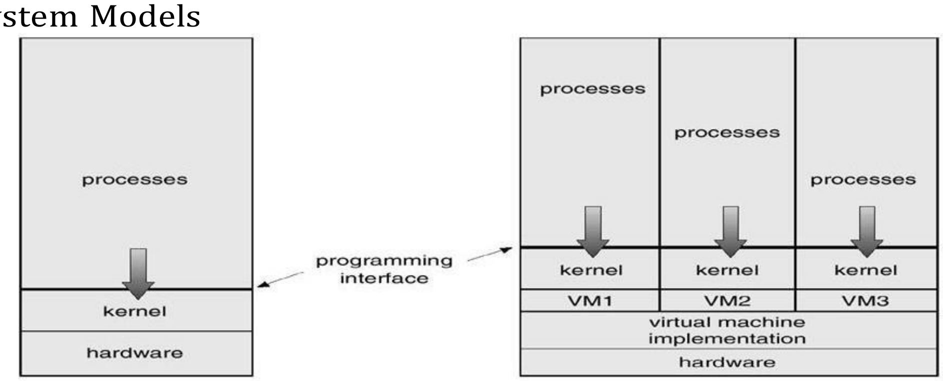
ii. Multiprocessor systems and clustered systems: Multiprocessor systems (also known as parallel systems or tightly coupled systems) are growing in importance. Such systems have two or more processors in close communication, sharing the computer bus and sometimes the clock, memory, and peripheral devices.

Clustered systems differ from multiprocessor systems, however, in that they are composed of two or more individual systems coupled together. Clustering is usually used to provide high-availability service; that is, service will continue even if one or more systems in the cluster fail. High availability is generally obtained by adding a level of redundancy in the system.

c). Explain with a neat diagram VM-Ware Architecture **8 Marks**

Answer:

- A *virtual machine* takes the layered approach to its logical conclusion. It treats hardware and theoperating system kernel as though they were all hardware.
- A virtual machine provides an interface *identical* to the underlying bare hardware.
- The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory.
- The resources of the physical computer are shared to create the virtual machines.
  - ✦ CPU scheduling can create the appearance that users have their own processor.
  - ✦ Spooling and a file system can provide virtual card readers and virtual line printers.
  - ✦ A normal user time-sharing terminal serves as the virtual machine operator's console.
  - System Models



Non-virtual Machine Virtual Machine

Advantages/Disadvantages of Virtual Machines

- The virtual-machine concept provides complete protection of system resources since each virtual
- Machine is isolated from all other virtual machines. This isolation, however, permits no direct sharing of resources.
- A virtual-machine system is a perfect vehicle for operating-systems research and development. System development is done on the virtual machine, instead of on a

physical machine and so does not disrupt normal system operation.

The virtual machine concept is difficult to implement due to the effort required to provide an *exact* duplicate to the underlying machine.

OR

2a. List& Explain the services of the operating system that are helpful for the user and the system. 6 Marks

Answer:

Operating systems can be explored from two viewpoints: the user and the system.

User View: The user's view of the computer varies according to the interface being used.

Most computer users sit in front of a PC, consisting of a monitor, keyboard, mouse, and system unit. Such a system is designed for one user to monopolize its resources. ***The goal is to maximize the work (or play) that the user is performing***. In this case, the operating system is designed mostly for ***ease of use***, with some attention paid to performance and none paid to ***resource utilization***-how various hardware and software resources are shared. Performance is, of course, important to the user; but rather than resource utilization, such systems are optimized for the single-user experience.

System View: From the computer's point of view, the operation system is the program most intimately involved with the hardware. In this context, we can view an operating system as a ***resource allocator***. A computer system has many resources that may be required to solve a problem: CPU time, memory space, file-storage space, I/O devices, and so on. The operating system acts as the manager of these resources.

A ***control program*** manages the execution of user programs to prevent errors and improper use of the computer. It is especially concerned with the operation and control of I/O devices.

Following are the six services provided by operating systems to the convenience of the users.

1. ***User interface***: Almost all operating systems have a user interface (UI). This interface can take several forms. One is a command-line interface (CLI) and other is a graphical user interface (GUI) is used.
2. ***Program Execution:*** The purpose of computer systems is to allow the user to execute programs. So the operating system provides an environment where the user can conveniently run programs.
3. ***I/O Operations:*** Each program requires an input and produces output. This involves the use of I/O. So the operating systems are providing I/O makes it convenient for the users to run programs.
4. ***File System Manipulation:*** The output of a program may need to be written into new files or input taken from some files. The operating system provides this service. Finally, some programs include permissions management to allow or deny access to files or directories based on file ownership.
5. ***Communications:*** The processes need to communicate with each other to exchange information during execution. It may be between processes running on the same computer or running on the different computers. Communications can be occur in two ways: (i) shared memory or (ii) message passing
6. ***Error Detection:*** An error is one part of the system may cause malfunctioning of the complete system. To avoid such a situation operating system constantly monitors the system for detecting the errors. This relieves the user of the worry of errors propagating to various part of the system and causing malfunctioning.

Following are the three services provided by operating systems for ensuring the efficient operation of the system itself.

1. **Resource allocation**
2. **Accounting**

Protection

2b. Explain the different computing environment.

Computing Environments

The different computing environments are -

Traditional Computing

** The current trend is toward providing more ways to access these computing environments.**

Web technologies are stretching the boundaries of traditional computing. Companies establish portals, which provide web accessibility to their internal servers. Network computers are essentially terminals that understand web-based computing. Handheld computers can synchronize with PCs to allow very portable use of company information. Handheld PDAs can also connect to wireless networks to use the company's web portal. The fast data connections are allowing home computers to serve up web pages and to use networks. Some homes even have firewalls to protect their networks.

** 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.**

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 file 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 a node has joined the network, it can begin providing services to—and requesting services from—other nodes in the network.

Determining what services are available is accomplished in one of two general ways:

 When a node joins a network, it registers its service with a centralized lookup service on the network. Any node desiring a specific service first contacts this centralized lookup service to determine which node provides the service. The remainder of the communication takes place between the client and the service provider.

 A peer acting as a client must know, which node provides a desired service by broadcasting a request for the service to all other nodes in the network. The node (or nodes) providing that service responds to the peer making the request. To support this approach, a discovery protocol must be provided that allows peers to discover services provided by other peers in the network.

Web-Based Computing

▯ Web computing has increased the importance on networking. Devices that were not previously networked now include wired or wireless access. Devices that were networked now have faster network connectivity.

▯ The implementation of web-based computing has given rise to new categories of devices, such as load balancers, which distribute network connections among a pool of similar servers.

Operating systems like Windows 95, which acted as web clients, have evolved into Linux and Windows XP, which can act as web servers as well as clients. Generally, the Web has increased the complexity of devices, because their users require them to be web-enabled.

▯ The design of an operating system is a major task. It is important that the goals of the new system be well defined before the design of OS begins. These goals form the basis for choices among various algorithms and strategies.

2c. What are system calls? Explain its types **8Marks**

Answer:

- System calls provide an interface between the process and the operating system.
- System calls allow user-level processes to request some services from the operating system which process itself is not allowed to do.
- For example, for I/O a process involves a system call telling the operating system to read or write particular area and this request is satisfied by the operating system.
- System calls can be grouped roughly into five major categories: process control, file manipulation, device manipulation, information maintenance, and communications.

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.

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.

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.

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 time 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.

Communication: There are two common models of interprocess 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.

MODULE-2

3a. What is Process and explain the state diagram 10Marks Answer:

Process: Program under execution, which is in main memory

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.

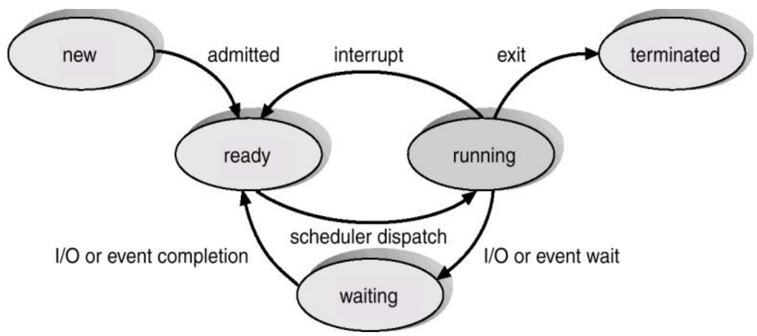


Figure: Diagram of process state.

Each process may be in one of the following states:

- **New State:** The process is being created.
- **Running State**: A process is said to be running if it has the CPU, that is, process actually using the CPU at that particular instant.
- **Blocked (or waiting) State:** A process is said to be blocked if it is waiting for some event to happen such that as an I/O completion before it can proceed. Note that a process is unable to run until some external event happens.
- **Ready State:** A process is said to be ready if it needs a CPU to execute. A ready state process is runnablebut temporarily stopped running to let another process run.
- **Terminated state:** The process has finished execution.

Process Control Block (PCB)

Each process is represented in the operating system by a process control block (PCB)-also called a task control block. A PCB is shown in Figure below. It contains many pieces of information associated with a specific process, including these:

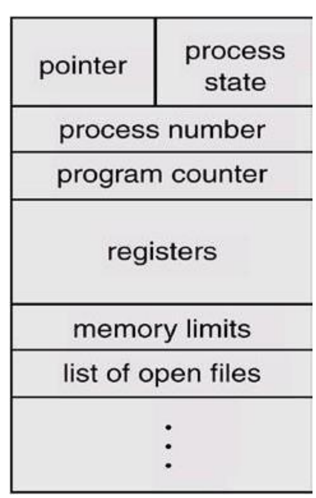


Figure: Process control block (PCB).

- Process state

- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information

Process state: The state may be new, ready, running, waiting, halted, and SO on.
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 numbers, job or process numbers, and so on.
Status information: The information includes the list of I/O devices allocated to this process, a list of open files, and so on.

3b What is inter-process communication? Discuss message passing and the shared memory concept of IPC. 10 Marks

Answer:

Concurrent execution of cooperating processes requires mechanisms that allow processes to communicate with one another and to synchronize their actions.
Cooperating processes require an interprocess communication (IPC) mechanism that will allow them to exchange data and information.
There are two fundamental models of interprocess communication:

(1) shared memory and
(2) message passing.

In the shared-memory model, a region of memory that is shared by cooperating processes is established. Processes can then exchange information by reading and writing data to the shared region.
In the message-passing model, communication takes place by means of messages exchanged between the cooperating processes. The two communications models are contrasted in Figure below.

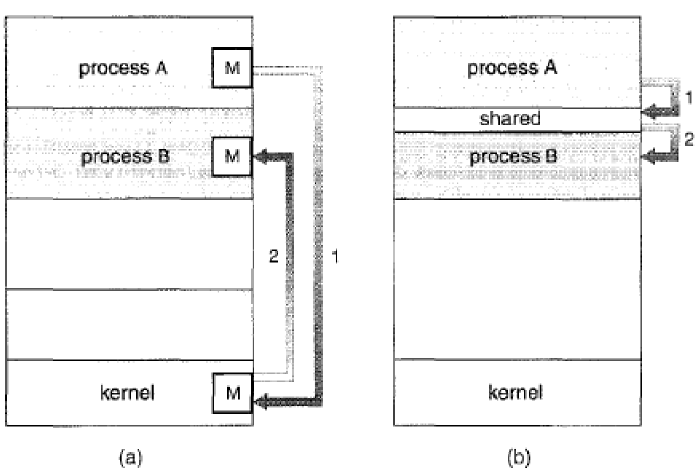


Figure: Communications models. (a) Message passing. (b) Shared memory.

Shared-Memory Systems

Interprocess communication using shared memory requires communicating processes to establish a region of shared memory. Typically, a shared-memory region resides in the address space of the process creating the shared-memory segment.

Other processes that wish to communicate using this shared-memory segment must attach it to their address space.

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.

Two types of buffers can be used. The unbounded buffer places no practical limit on the size of the buffer. The consumer may have to wait for new items, but the producer can always produce new items.

The bounded buffer assumes a fixed buffer size. In this case, the consumer must wait if the buffer is empty, and the producer must wait if the buffer is full.

Message-Passing Systems

Message passing provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space and is particularly useful in a distributed environment, where the communicating processes may reside on different computers connected by a network.

A message-passing facility provides at least two operations: send (message) and receive (message). Messages sent by a process can be of either fixed or variable size. If only fixed-sized messages can be sent, the system-level implementation is straightforward. This restriction, however, makes the task of programming more difficult.

If processes *P* and Q want to communicate, they must send messages to and receive messages from each other; a **communication link** must exist between them.

Here are several methods for logically implementing a link and the send() / receive() operations:

- Direct or indirect communication
- Synchronous or asynchronous communication
- Automatic or explicit buffering

OR

4. (a). explain detail the multithreading model **6 Marks**

Answer:

Many-to-One Model

The many-to-one model (Figure below) maps many user-level threads to one kernel thread. Thread management is done by the thread library in user space, so it is efficient; but the entire process will block if a thread makes a blocking system call.

Also, because only one thread can access the kernel at a time, multiple threads are unable to run in parallel on multiprocessors. **Green threads**-a thread library available for Solaris-uses this model, as does **GNU Portable Threads**.

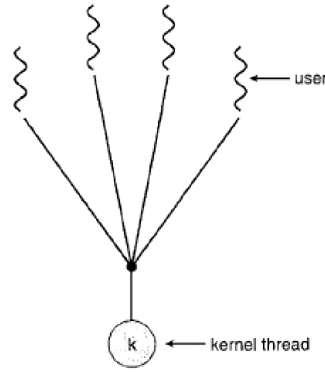


Figure: Many-to-one model.

One-to-One Model

The one-to-one model (Figure below) maps each user thread to a kernel thread. It provides more concurrency than the many-to-one model by allowing another thread to run when a thread makes a blocking system call; it also allows multiple threads to run in parallel on multiprocessors. The only drawback to this model is that creating a user thread requires creating the corresponding kernel thread.

Because the overhead of creating kernel threads can burden the performance of an application, most implementations of this model restrict the number of threads supported by the system. Linux, along with the family of Windows operating systems-including Windows 95, 98, NT, 2000, and XP implement the one-to-one model.

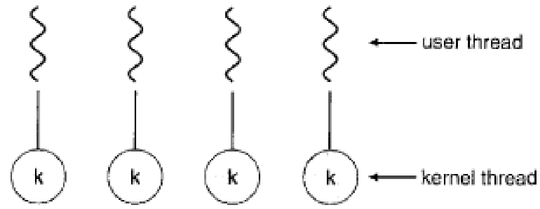


Figure: One-to-one model.

Many-to-Many Model

The many-to-many model (Figure below) multiplexes many user-level threads to a smaller or equal number of kernel threads. The number of kernel threads may be specific to either a particular application or a particular machine (an application may be allocated more kernel threads on a multiprocessor than on a uniprocessor).

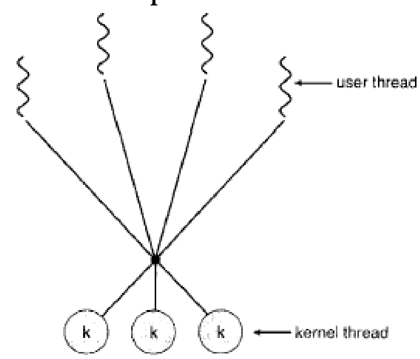


Figure: Many-to-many model.

One popular variation on the many-to-many model still multiplexes many user-level threads to a smaller or equal number of kernel threads but also allows a user-level thread to be bound to a kernel thread. This variation, sometimes referred to as the *two-level model* (Figure below), is supported by operating systems such as IRIX, HP-UX, and Tru64 UNIX.

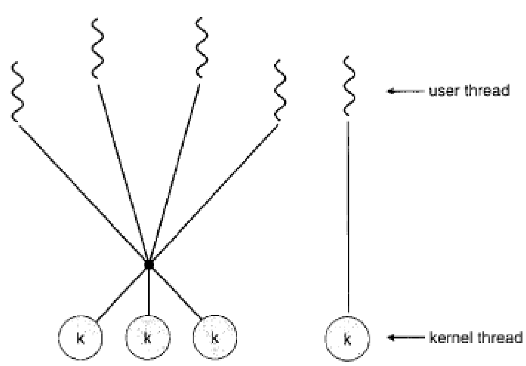


Figure: Two-level model.

(c). Calculate average waiting and average turnaround times by drawing the Gantt chart using FCFS, SJF and RR (q=4ms) and priority scheduling(Higher nUmber is highest priority) 8 Marks

Process	B.T(ms)	Priority
P1	24	1
P2	03	2
P3	03	3

1. First Come First Serve (FCFS)

Gantt Chart:

CopyEdit

| P1 | P2 | P3 |

0 24 27 30

Turnaround Time (TAT) = Completion Time - Arrival Time
Waiting Time (WT) = Turnaround Time - Burst Time

Process	BT	CT	TAT = CT - AT	WT = TAT - BT
P1	24	24	24	0
P2	3	27	27	24
P3	3	30	30	27

AWT = (0 + 24 + 27) / 3 = 17 ms
ATAT = (24 + 27 + 30) / 3 = 27 ms

---

2. Shortest Job First (SJF) - Non-Preemptive

Order of Execution: P2 → P3 → P1

Gantt Chart:

CopyEdit

| P2 | P3 | P1 |

0 3 6 30

Process	BT	CT	TAT = CT - AT	WT = TAT - BT
P2	3	3	3	0
P3	3	6	6	3
P1	24	30	30	6

AWT = (0 + 3 + 6) / 3 = 3 ms
ATAT = (3 + 6 + 30) / 3 = 13 ms

---

3. Round Robin (RR) - Quantum = 4 ms

Order of Execution: P1(4) → P2(3) → P3(3) → P1(4) → P1(4) → P1(4) → P1(4)

Gantt Chart:

CopyEdit

| P1 | P2 | P3 | P1 | P1 | P1 | P1 |

0 4 7 10 14 18 22 26 30

Process	BT	CT	TAT = CT - AT	WT = TAT - BT
P1	24	30	30	6
P2	3	7	7	4
P3	3	10	10	7

AWT = (6 + 4 + 7) / 3 = 5.67 ms
ATAT = (30 + 7 + 10) / 3 = 15.67 ms

---

4. Priority Scheduling (Higher Number = Higher Priority) - Non-Preemptive

Order of Execution: P3 → P2 → P1

Gantt Chart:

CopyEdit

| P3 | P2 | P1 |

0   3   6   30

Process	BT	Priority	CT	TAT = CT - AT	WT = TAT - BT
P3	3	3	3	3	0
P2	3	2	6	6	3
P1	24	1	30	30	6

AWT = (0 + 3 + 6) / 3 = 3 ms
ATAT = (3 + 6 + 30) / 3 = 13 ms

Final Results Comparison

Algorithm	AWT (ms)	ATAT (ms)
FCFS	17	27
SJF	3	13
RR (q=4)	5.67	15.67
Priority	3	13

Conclusion:

- **SJF and Priority Scheduling provide the best results with the lowest AWT (3 ms) and ATAT (13 ms).**
- **FCFS has the worst AWT and ATAT due to long waiting times.**
- **Round Robin balances fairness but increases waiting time compared to SJF and Priority.**

MODULE-3

5a. Critical Section and Peterson probel. **5 Marks**

Answer:

A classic software-based solution to the critical-section problem known as Peterson's solution.

Peterson's solution is restricted to two processes that alternate execution between their critical sections and remainder sections. The processes are numbered P_0 and P_1.

For convenience, when presenting *Pi,* we use *Pj* to denote the other process; that is, j equals 1 - i. Peterson's solution requires two data items to be shared between the two processes:

```
int turn;
boolean flag[2];
```

The variable turn indicates whose turn it is to enter its critical section. That is, if turn == i, then process P_i is allowed to execute in its critical section. The flag array is used to indicate if a process *is ready* to enter its critical section.

For example, if flag [i] is true, this value indicates that P_i is ready to enter its critical section. With an explanation of these data structures complete, we are now ready to describe the algorithm shown in Figure below.

```
do {

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

        critical section

    flag[i] = FALSE;

        remainder section

} while (TRUE);
```

Figure: The structure of process P_i in Peterson's solution.

To enter the critical section, process P_i first sets flag [i] to be true and then sets turn to the value j, thereby asserting that if the other process wishes to enter the critical section it can do so. If both processes try to enter at the same time, turn will be set to both i and j at roughly the same time. Only one of these assignments will last; the other will occur, but will be overwritten immediately.

To prove property ***Mutual exclusion is preserved***, we note that each P_i enters its critical section only if either flag [j] == false or turn == i.

Also note that, if both processes can be executing in their critical sections at the same time, then flag [i] ==flag [j] == true. These two observations imply that P_0 and P_1 could not have successfully executed their while statements at about the same time, since the value of turn can be either 0 or 1, but cannot be both.

5 c. What is a semaphore? Classical Dining Philosopher problem gives a solution using semaphore. 8 Marks

Answer:

A **semaphore** S is an integer variable that, apart from initialization, is accessed only through two standard atomic operations: **wait** and **signal**. These operations were originally termed P (for wait; from the Dutch proberen, to test) and V (for signal; from verhogen, to increment). The classical definition of wait in pseudo code is

```
wait(S)
{
    while (S <= 0)
    ; // no-op
    S --;
}
```

The classical definitions of signal in pseudo code is

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

The dining-philosophers problem is considered a classic synchronization problem neither because of its practical importance nor because computer scientists dislike philosophers but because it is an example of a large class of concurrency-control problems. It is a simple representation of the need to allocate several resources among several processes in a deadlock-free and starvation-free manner.

One simple solution is to represent each chopstick with a semaphore. A philosopher tries to grab a chopstick by executing a wait () operation on that semaphore; she releases her chopsticks by executing the signal () operation on the appropriate semaphores.

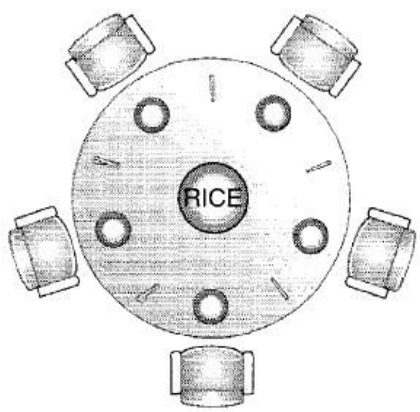


Figure 1 The situation of the dining philosophers.

Thus, the shared data are

semaphore chopstick [5] ;

where all the elements of chopstick are initialized to 1. The structure of philosopher *i* is shown in Figure 2.

Although this solution guarantees that no two neighbors are eating simultaneously, it nevertheless must be rejected because it could create a deadlock. Suppose that all five philosophers become hungry simultaneously and each grabs her left chopstick. All the elements of chopstick will now be equal to O. When each philosopher tries to grab her right chopstick, she will be delayed forever.

```
do {
```

```
wait (chopstick[i] );
wait(chopstick[(i+1) % 5]);
. . . . . . . . . .
// eat
. . . . . . . . . .
signal (chopstick[i] );
signal (chopstick[(i+1) % 5]);
. . . . . . . . . .
// think
. . . . . . . . . .
} while (TRUE);
```

Figure 2 The structure of philosopher i.

A solution to the dining-philosophers problem that ensures freedom from deadlocks.

- Allow at most four philosophers to be sitting simultaneously at the table.
- Allow a philosopher to pick up her chopsticks only if both chopsticks are available (to do this she must pick them up in a critical section).
- Use an asymmetric solution; that is, an odd philosopher picks up first her left chopstick and then her right chopstick, whereas an even philosopher picks up her right chopstick and then her left chopstick.

Finally, any satisfactory solution to the dining-philosophers problem must guard against the possibility that one of the philosophers will starve to death. A deadlock-free solution does not necessarily eliminate the possibility of starvation.

OR

6.(a). Define deadlock. What are the necessary conditions for deadlock to occur?

5 Marks

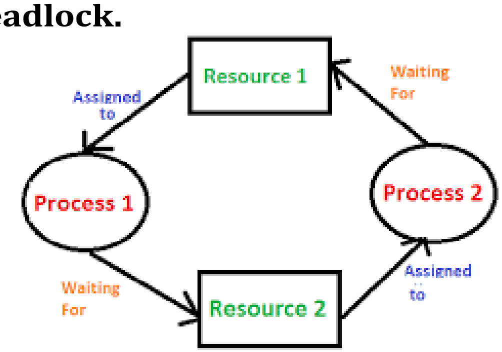
Answer:

In a multiprogramming environment, several processes may compete for a **finite number of resources**.

A process requests **resources; and if the resources are not available at that time, the process enters a waiting state**.

Sometimes, **a waiting process is never again able to change state**, because the resources it has **requested are held by other waiting process**es.

This situation is called a **deadlock.**



In a deadlock, processes never finish executing, and system resources are tied up, preventing other jobs from starting.

Necessary Conditions

A deadlock situation can arise if the following four conditions hold simultaneously in a system:

1. **Mutual Exclusion Condition**: The resources involved are non-shareable.
Explanation: At least one resource must be held in a non-shareable mode, that is, only one process at a time claims exclusive control of the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.

2. **Hold and Wait Condition**: Requesting process hold already the resources while waiting for requested resources.
Explanation: There must exist a process that is holding a resource already allocated to it while waiting for additional resource that are currently being held by other processes.

3. **No-Preemptive Condition**: Resources already allocated to a process cannot be preempted. Explanation: Resources cannot be removed from the processes are used to completion or released voluntarily by the process holding it.

4. **Circular Wait Condition**: The processes in the system form a circular list or chain where each process in the list is waiting for a resource held by the next process in the list. There exists a set {P0, P1, …, P0} of waiting processes such that P0 is waiting for a resource that is held by P1, P1 is waiting for a resource that is held by P2, …, Pn−1 is waiting for a resource that is held by Pn, and P0 is waiting for a resource that is held by P0.

6 b. Consider the following snapshot of a system:

	Allocation				***Max***				***Available***			
	A	***B***	***C***	***D***	***A***	***B***	***C***	***D***	***A***	***B***	***C***	***D***
P0	0	0	1	2	0	0	1	2	1	5	2	0
P1	1	0	0	0	1	7	5	0				
P2	1	3	5	4	2	3	5	6				
P3	0	6	3	2	0	6	5	2				
P4	0	0	1	4	0	6	5	6				

Answer the following questions using the banker's algorithm:

a. What is the content of the matrix *Need?*

b. Is the system in a safe state?

c. If a request from process PI arrives for (0, 4, 2, 0), can the request be granted immediately?

9 Marks

Answer:

a. Since *Need* = *Max* − *Allocation*, the content of *Need* is

A B CD
0 0 0 0

0 7 5 0
1 0 0 2
0 0 2 0
0 6 4 2

b. Yes, the sequence <*P*0, *P*2, *P*1, *P*3, *P*4> satisfies the safety requirement.

c. Yes. Since

i. (0,4,2,0) _ *Available* = (1,5,2,0)

ii. (0,4,2,0) _ *Maxi* = (1,7,5,0)

iii. The new system state after the allocation is made is

	Allocation	***Max***	***Need***	***Available***
	A B C D	***A B C D***	***A B C D***	***A B C D***
*P*0	0 0 1 2	0 0 1 2	0 0 0 0	1 1 0 0
*P*1	1 4 2 0	1 7 5 0	0 3 3 0	
*P*2	1 3 5 4	2 3 5 6	1 0 0 2	
*P*3	0 6 3 2	0 6 5 2	0 0 2 0	
*P*4	0 0 1 4	0 6 5 6	0 6 4 2	

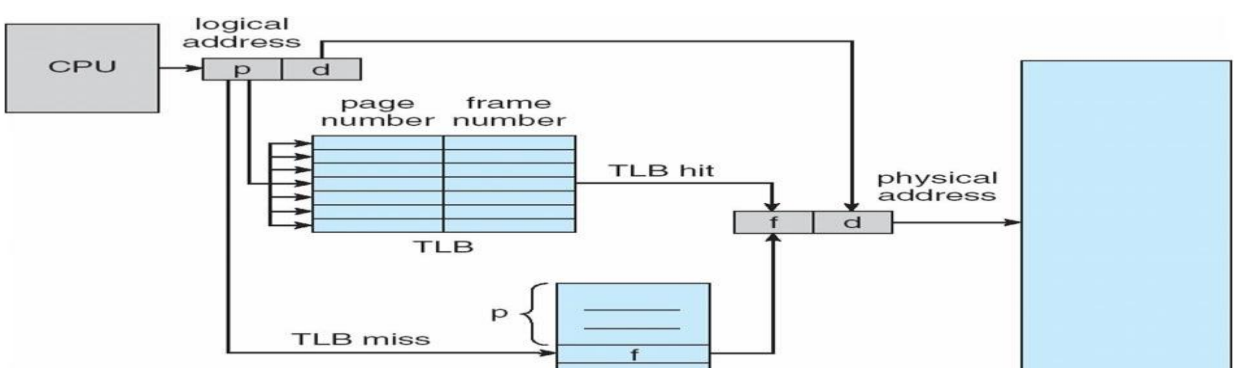
and the sequence < *P*0, *P*2, *P*1, *P*3, *P*4 > satisfies the safety requirement.

MODULE-4

7. (a). What is TLB? Explain TLB in detail with a paging system with a neat diagram. **6 Marks**

Answer:

- The standard solution to this problem is to use a special, small, fast lookup hardware cache, called a translation look-aside buffer (TLB).
- The TLB is associative, high-speed memory. Each entry in the TLB consists of two parts: **a key (or tag) and a value.**
- When the associative memory is presented with an item, the item is compared with all keys simultaneously.
- The TLB is used with page tables in the following way. The TLB contains only a few of the page-table entries. When a logical address is generated by the CPU, its page number is presented to the TLB.
- If the page number is found, its frame number is immediately available and is used to access memory.
- The whole task may take less than 10 percent longer than it would if an unmapped memory reference were used.
- If the page number is not in the TLB (known as a TLB miss), a memory reference to the page table must be made. When the frame number is obtained, we can use it to access memory (Figure below).



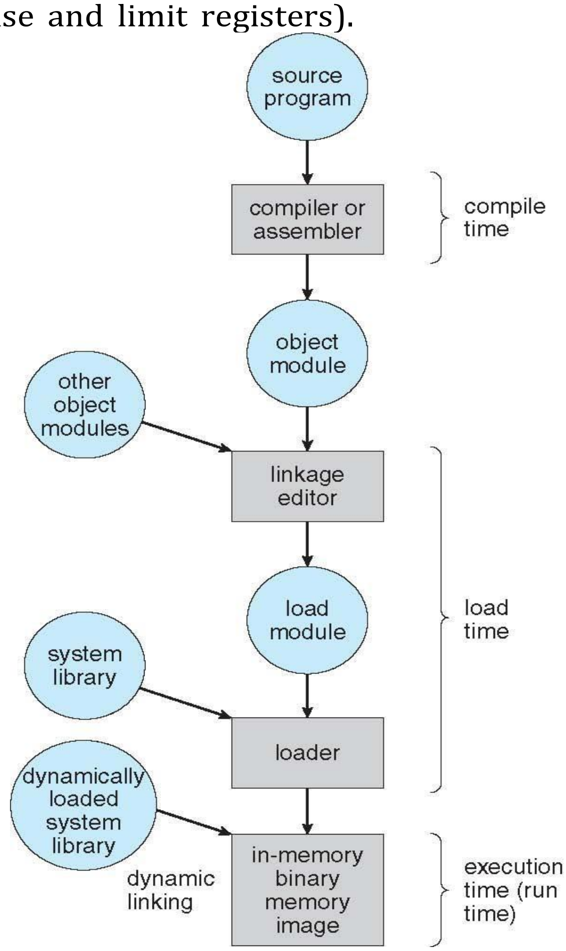
(b). With the help of a neat diagram, explain the various steps of address binding.

6 Marks

Answer:

Address binding of instructions and data to memory addresses can happen at three different stages.

1. **Compile time:** The compile time is the time taken to compile the program or source code. During compilation, if memory location known a priori, then it generates absolute codes.
2. Load time: It is the time taken to link all related program file and load into the main memory. It must generate relocatable code if memory location is not known at compile time.
3. Execution time: It is the time taken to execute the program in main memory by processor. Binding delayed until run time if the process can be moved during its execution from one memory segment to another. Need hardware support for address maps (e.g., base and limit registers).



(c). Consider the page reference string: 1,0,7,1,0,2,1,2,3,0,3,2,4,0,3,6,2,1 for a memory with three frames. Determine the number of page faults using the FIFO, Optimal, and LRU replacement algorithms. Which algorithm is most efficient?

8 Marks

Answer:

1. FIFO (First-In, First-Out)

FIFO replaces the oldest page in the frame.

Page	Frame 1	Frame 2	Frame 3	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	1	2	3	Yes
2	1	2	3	No
0	1	0	3	Yes
1	1	0	3	No
7	7	0	3	Yes
0	7	0	3	No
1	7	1	3	Yes

Export to Sheets

Total Page Faults (FIFO): 15

2. Optimal Page Replacement

Optimal replacement replaces the page that will not be used for the longest time in the future.

Page	Frame 1	Frame 2	Frame 3	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	1	2	3	Yes
2	1	2	3	No
0	1	0	3	Yes
1	1	0	3	No
7	7	0	3	Yes
0	7	0	3	No
1	7	1	3	Yes

Export to Sheets

Total Page Faults (Optimal): 12

3. LRU (Least Recently Used)

LRU replaces the page that has been used the least recently.

Page	Frame 1	Frame 2	Frame 3	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	1	2	3	Yes
2	1	2	3	No
0	1	0	3	Yes
1	1	0	3	No
7	7	0	3	Yes
0	7	0	3	No
1	7	1	3	Yes

Export to Sheets

Total Page Faults (LRU): 12

Efficiency Comparison

The efficiency of a page replacement algorithm is determined by the number of page faults it incurs – the lower the page faults, the more efficient the algorithm.

In this case:

- **Optimal and LRU are more efficient than FIFO as they both result in fewer page faults (12) compared to FIFO (15).**
- **Optimal and LRU have the same number of page faults for this particular reference string.**

Important Note:

- **The efficiency of page replacement algorithms can vary depending on the specific reference string.**
- **The Optimal algorithm is theoretically the most efficient, but it's not practically implementable because it requires future knowledge of the reference string. It serves as a benchmark to compare other algorithms.**
- **LRU often performs very close to Optimal in practice and is a commonly used algorithm.**

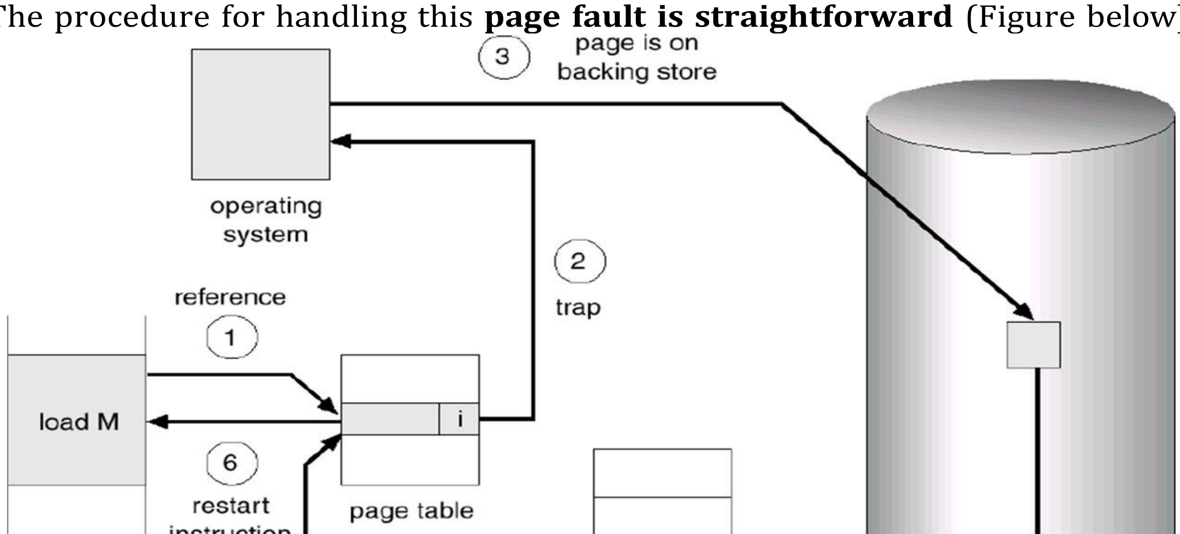
Sources and related content

OR

8. (a). What is Page Fault? Explain the steps in handling page faults using the appropriate diagram.

Answer: **10 Marks**

- Consider how an executable **program might be loaded from disk into memory**.
- Loading the **entire program into memory results in loading the executable code for *all* options**, regardless of whether an option is ultimately selected by the user or not.
- An alternative strategy is to **initially load pages only** as they are needed.
- This technique is known as **demand paging and is commonly used in virtual memory systems**.
- With demand-paged **virtual memory, pages are only loaded when they are demanded during program execution**; pages that are never accessed are thus **never loaded into physical memory**.
  - The paging hardware, in **translating the address through the page table**, will notice that the **invalid bit is set, causing a trap to the operating system**.
  - This trap is the result of the **operating system's failure to bring the desired page into memory**.
  - The procedure for handling this **page fault is straightforward** (Figure below):



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

A page fault causes the following sequence to occur:

1. **Trap** to the operating system.
2. Save the **user registers and process state**.
3. Determine that the **interrupt was a page fault**.
4. Check that the **page reference was legal and determine the location of the page on the disk**.
5. Issue a read from the **disk to a free frame**:

a. ·Wait in a queue for this device **until the read request is serviced**.

b. Wait for the device **seek and / or latency time**.

c. Begin the **transfer of the page to a free frame**.

6. While waiting, allocate the CPU to some other user (CPU scheduling, optional).

9 (b)Explain File Allocation methods 10

ALLOCATIONMETHODS

Allocation methods address the problem of allocating space to files so that disk space is utilizedeffectivelyandfilescanbeaccessedquickly.

Three methodsexistforallocatingdiskspace

 Contiguousallocation

 Linkedallocation

 Indexedallocation

Contiguousallocation:

 Requiresthateachfile occupyasetofcontiguousblocks onthedisk

 Accessing a file is easy – only need the starting location (block #) and length (number ofblocks)

 Contiguous allocation of a file is defined by the disk address and length (in block units) ofthe first block. If the file is n blocks long and starts at location b, then it occupies blocks b,b + 1, b + 2, ... ,b + n - 1. The directory entry for each file indicates the address of thestartingblockandthelengthofthearea allocatedforthisfile.

 Accessingafilethathasbeenallocatedcontiguouslyiseasy.Forsequentialaccess,thefile system

remembers the disk address of the last block referenced and when necessary,reads the next block. For direct access to block i of a file that starts at block b, we canimmediately access block b + i. Thus, both sequential and direct access can be supportedbycontiguousallocation.

Disadvantages:

1. Finding space for a new file is difficult. The system chosen to manage free spacedetermines how this task is accomplished. Any management system can be used, butsomeareslowerthanothers.
2. Satisfying a request of size n from a list of free holes is a problem. First fit and bestfitarethemostcommonstrategiesusedtoselectafreehole fromthesetofavailable

holes.
3. Theabovealgorithmssufferfromtheproblem ofexternalfragmentation.
- Asfilesareallocatedanddeleted,thefreediskspaceisbrokenintopieces.
- Externalfragmentation existswheneverfreespaceisbrokenintochunks.
- It becomes a problem when the largest contiguous chunk is insufficient for arequest; storage is fragmented into a number of holes, none of which is largeenoughtostore thedata.
- Dependingonthetotalamountofdiskstorageandtheaveragefilesize,externalfragmentationmaybeaminor oramajorproblem.

LinkedAllocation:
- Solvestheproblemsofcontiguous allocation
- Eachfileisalinked list ofdiskblocks:blocksmaybescatteredanywhereonthedisk
- Thedirectorycontainsapointertothefirstandlastblocksofafile
- Creatinganewfile requiresonlycreationofanew entryinthedirectory
- Writingto afilecausesthefree-spacemanagementsystemtofindafreeblock
- Thisnewblockis writtentoandislinked totheendofthefile
- Readingfromafilerequiresonlyreadingblocksbyfollowingthepointersfromblocktoblock.

Advantages
- Thereisno externalfragmentation
- Anyfreeblocksonthe freelistcanbeusedtosatisfyarequestfordiskspace
- Thesizeofafileneed notbedeclaredwhenthefileiscreated
- Afilecancontinuetogrowaslongasfreeblocksareavailable
- Itisnevernecessarytocompactdiskspaceforthesakeoflinked allocation(however,fileaccessefficiencymayrequireit)

- Each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on thedisk.Thedirectorycontainsapointertothefirstandlastblocksofthefile.
- For example, a file of five blocks might start at block 9 and continue at block 16, thenblock 1, then block 10, and finally block 25. Each block contains a pointer to the nextblock. These pointers are not made available to the user. A disk address (the pointer)requires4bytesinthedisk.
- Tocreateanewfile,wesimplycreateanewentryilethedirectory.Withlinkedallocation, each directory entry has a pointer to the first disk block of the file. This pointeris initialized to nil (the end-of-list pointer value) to signify an empty file. The size field isalsosetto0.
- A write to the file causes the free-space management system to filed a free block, and thisnewblockiswrittentoandislinkedtothe endofthefile.
- To read a file, we simply read blocks by following the pointersfromblock to block.There

is no external fragmentation with linked allocation, and any free block on the free-spacelistcanbeusedtosatisfyarequest.Thesizeofafileneednotbedeclaredwhenthatfileiscre ate

d.
 A file can continue to grow as long as free blocks are available. Consequently, it is nevernecessarytocompactdiskspace.
 Disadvantages:

1. The major problem is that it can be used effectively only for sequential-accessfiles. To filed the i th block of a file, we must start at the beginning of that

fileandfollowthepointersuntilwegettotheithblock.
2. Space required for the pointers. Solution is clusters. Collect blocks into multiplesandallocateclustersrather thanblocks
3. Reliability - the files are linked together by pointers scattered all over the diskandifapointerwerelostordamagedthenallthelinksarelost.

FileAllocationTable:

Asectionofdiskatthebeginningofeachvolumeissetasidetocontainthetable.Thetablehasone ent
ryforeachdiskblockandisindexedbyblocknumber.

TheFATisusedinmuchthesamewayasalinkedlist.Thedirectoryentrycontainstheblocknum be
rofthefirstblockofthefile.
 Thetableentryindexedbythat blocknumbercontainstheblock numberofthe next blockinthefile.

 Thechaincontinuesuntilitreachesthelastblock,whichhasaspecialend-of-filevalueasthetableentry.

 Anunusedblockisindicatedbyatablevalueof 0.
 ConsideraFATwitha fileconsistingof diskblocks217, 618,and339.

Indexedallocation:
 Bringsallthepointerstogetherintoonelocationcalledindexblock.
 Eachfilehasitsownindexblock,whichisanarrayofdisk-blockaddresses.
 The ith entry in the index block points to theith block of the file. The directory containsthe
address of the index block. To find and read the ith block, we use the pointer in the ithindex-block entry.
 When the file is created, all pointers in the index block are set to nil. When the ith block

isfirstwritten,ablockisobtainedfromthefree-spacemanageranditsaddressisputintheithindex-block entry.

 Indexed allocation supports direct access, without suffering from external fragmentation,becauseanyfreeblockonthediskcansatisfyarequestformorespace.
 Disadvantages:
 Suffersfromsomeofthesameperformanceproblemsas linkedallocation
 Indexblockscanbecachedinmemory;however,datablocksmaybespreadalloverthediskv olume.
 Indexedallocationdoessufferfromwastedspace.
 Thepointeroverheadoftheindexblockisgenerallygreaterthanthepointeroverhea doflinkedallocation.

If the index block is too small, however, it will not be able to hold enough pointers for a large
file,and a mechanism will have to be available to deal with this issue. Mechanisms for this purposeincludethe following:
a) Linked scheme. An index block is normally one disk block. Thus, it can be read and writtendirectly
byitself.Toallowforlargefiles,wecanlinktogetherseveralindexblocks.Forexample,
an index block might contain a small header giving the name of the file and a set of thefirst 100
disk-block addresses. The next address (the last word in the index block) is nil (for asmallfile)orisapointertoanotherindexblock(foralargefile).
b) Multilevel index. A variant of linked representation uses a first-level index block to point to
aset of second-level index blocks, which in turn point to the file blocks. To access a block, theoperating system uses the first-level index to find a second-level index block and then uses
thatblock to find the desired data block. This approach could be continued to a third or fourth
level,dependingonthedesiredmaximumfilesize

c) Combined scheme. For eg. 15 pointers of the index block is maintained in the file's i node.The first 12 of these pointers point to direct blocks; that is, they contain addresses of
blocks thatcontain data of the file. Thus, the data for small files (of no more than 12 blocks) do
not need aseparate index block.If the block size is 4 KB,then up to 48KB of data can be accesseddirectly. The next three pointers point to indirect blocks. The first points to a single
indirectblock, which is an index block containing not data but the addresses of blocks that do
containdata. The second points to a double indirect block, which contains the address of a block
thatcontains the addresses of blocks that contain pointers to the actual data blocks. The last
pointercontainstheaddress ofa tripleindirectblock.

Performance

 Contiguousallocationrequiresonlyoneaccesstogetadiskblock.Sincewecaneasilykeep the initial address of the file in memory, we can calculate immediately the diskaddressofthe ithblockandreaditdirectly.

 For linked allocation, we can also keep the address of the next block in memory andread it directly. This method is fine for sequential access. Linked allocation shouldnotbeusedforanapplicationrequiringdirectaccess.

 Indexed allocation is more complex. If the index block is already in memory, then theaccess can bemadedirectly.However,keeping the index block inmemory requiresconsiderable space. If this memory space is not available, then we may have to read firsttheindexblockandthenthedesireddatablock.

FreeSpaceManagement

Thespacecreatedafterdeletingthefilescanbereused.Anotherimportantaspectofdiskmanagement is
keeping track of free space in memory. The list which keeps track of free space inmemory is called
the free-space list. To create a file, search the free-space list for the requiredamount of space and
allocate that space to the new file. This space is then removed from the free-space list. When a file
is deleted, its disk space is added to the free-space list. The free-space list, isimplementedindifferentwaysasexplainedbelow.

a) BitVector

 Fastalgorithmsexistforquicklyfindingcontiguousblocksofagivensize

 One simple approach is to use a bit vector, in which each bit represents a disk block,setto1iffreeor0ifallocated.

Forexample,consideradiskwhereblocks2,3,4,5,8,9,10,11,12,13,17and18arefree,andtherestoftheblock
sareallocated.Thefree-spacebitmapwouldbe
0011110011111100011

 Easytoimplementandalsoveryefficientinfindingthefirstfreeblockor‘n’ consecutivefreeblocks onthedisk.

 Thedown sideisthata 40GBdiskrequiresover5MBjusttostorethebitmap.

b) LinkedList

a. Alinkedlistcanalsobeusedtokeeptrack of allfreeblocks.

b. Traversing the list and/or finding a contiguous block of a given size are not easy, butfortunately are not frequently needed operations. Generally the system just adds andremovessingleblocksfromthebeginning ofthelist.

c. TheFATtablekeepstrackofthefreelistasjustone morelinked listonthetable.

c) Grouping

a. A variation on linked list free lists. It stores the addresses of n free blocks in

the firstfreeblock.Thefirstn-1blocksareactuallyfree.Thelastblockcontainstheaddressesofanothernfree

blocks, andsoon.

b. Theaddress of alargenumberoffreeblocks canbefoundquickly.
d) Counting
a. When therearemultiple contiguous blocks of freespacethen the system cankeeptrackofthestartingaddressofthegroupandthenumberofcontiguousfreeblocks.
b. Ratherthankeepingallistofnfreediskaddresses,wecankeeptheaddressoffirstfreeblockandthenumberoffreecontiguousblocksthatfollowthefirstblock.
c. Thustheoverallspaceisshortened.Itissimilartotheextentmethodofallocatingblocks.
e) SpaceMaps
a. Sun's ZFS file system was designed for huge numbers and sizes of files, directories,andeven file systems.
b. The resulting data structures could be inefficient if not implemented carefully.Forexample, freeing up a 1 GB file on a 1 TB file system could involve updatingthousandsof blocksof free listbitmapsif the filewasspreadacrossthedisk.
c. ZFSusesacombinationoftechniques,startingwithdividingthediskupinto(hundreds of)Metaslabsofamanageablesize,eachhavingtheirownspacemap.
d. Free blocks are managed using the counting technique, but rather than write theinformation to a table, it is recorded in a log-structured transaction record. Adjacentfreeblocksarealsocoalescedintoalargersinglefreeblock.

e. Anin-memoryspacemapisconstructedusingabalancedtreedatastructure,constructedfrom the log data.
f. The combination of the in-memory tree and the on-disk log provide for very fast andefficientmanagementoftheseverylargefilesandfreeblocks.

MODULE-5

9. (a). What is a file? What are its attributes? Explain file operations. 10 Marks Answer:

A **file is a named collection** of related information that is recorded on secondary storage. The information in a file is **defined by its creator**. Many different types of information may be stored in a **file-source programs**, **object programs**, executable programs, numeric data, text payroll records, graphic images, sound recordings, and so on.

File Attributes:

A file is named, for the convenience of its human users, and is referred to by its name.
A file's attributes vary from one operating system to another but typically consist of these:
• **Name**. The symbolic file name is the only information kept in human readable form.
• **Identifier.** This 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 systems that support different types of files.
• **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 (in bytes, words, or blocks) 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, and user identification**. 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:
The operating system can provide system calls to create, write, read, reposition, delete, and truncate files.

- **Creating a file.** Two steps are necessary 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 (in memory) the next block of the file should be put.
- **Repositioning within a file**. The directory is searched for the appropriate entry, and the **current-file-position pointer is repositioned to a given value, Repositioning within a file need not involve any actual I/O**. This file operation is also known as a file *seek.*
- **Deleting a file**. To delete a file, we search the **directory for the named file. Having found the associated directory entry, we release all file space**, so that it can be reused by other files, and erase the directory entry.
- **Truncating a file**. The user may **want to erase the contents of a file but keep its attributes**. Rather than forcing the user to delete the file and then **recreate it, this function allows all attributes to remain unchanged-except** for file length-but lets the file be reset to length zero and its file space released.

(or)

10 A. Explain the access matrix method of system protection with the domain as objects and its implementation. 6 Marks

Answer:

Access Matrix

- The model of protection can be viewed **abstractly as a matrix**, called an **access matrix**.
- The rows of the **access matrix represent domains**, and the columns **represent objects**. Each entry in the **matrix consists of a set of access rights**.
- The **entry access(i,j)** defines the set of operations that a process executing in **domain D_i** can invoke on **object O_j.**
- To illustrate these concepts, we consider the access matrix shown in Figure below.
- There are **four domains and four objects-three files (F_1, F_2, F_3)** and one laser printer. A process executing in domain D_1 can read files **F_1 and F_3 .**
- A process executing in domain **D_4** has the same privileges as one executing in domain **D_1**; but in addition, it can also write onto files **F_1 and F_3.**
- Note that the **laser printer** can be accessed **only by a process** executing in **domain D_2.**

object / domain	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	

- ❑ The **access-matrix** scheme provides us with the mechanism for **specifying a variety of policies**.
- ❑ The mechanism consists of **implementing the access matrix and ensuring** that the semantic properties ***we* have outlined indeed hold**.
- ❑ More **specifically**, we must ensure that a **process executing in domain D_i can access only those objects specified in row i, and then only as allowed by the access-matrix entries**.

Implementation of Access Matrix

How can the access matrix be implemented effectively? In general the matrix will be sparse; that is, most of the entries will be empty. Although data structure techniques are available for representing sparse matrices, they are not particularly useful for this application, because of the way in which the protection facility is used.

Methods:

- Global Table
- Access Lists for Objects
- Capability Lists for Domains
- A Lock-Key Mechanism

- **10 B. Cylinder Range: 0 to 4999**
- **Current Head Position: 143**
- **Previous Head Position: 125 (Not directly relevant for calculations, but shows direction)**
- **Request Queue (FIFO Order): 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130**

1. FCFS (First-Come, First-Served)

FCFS processes requests in the order they arrive.

- **143 -> 86 -> 1470 -> 913 -> 1774 -> 948 -> 1509 -> 1022 -> 1750 -> 130**
- **Distance: |143-86| + |86-1470| + |1470-913| + |913-1774| + |1774-948| + |948-1509| + |1509-1022| + |1022-1750| + |1750-130|**
- **Total Distance: 6408**

2. SSTF (Shortest Seek Time First)

SSTF selects the request closest to the current head position.

- **143 -> 130 -> 86 -> 913 -> 948 -> 1022 -> 1470 -> 1509 -> 1750 -> 1774**
- **Distance: |143-130| + |130-86| + |86-913| + |913-948| + |948-1022| + |1022-1470| + |1470-1509| + |1509-1750| + |1750-1774|**
- **Total Distance: 2363**

3. SCAN (Elevator Algorithm)

SCAN moves the head in one direction, servicing requests along the way, then reverses direction. We need to know the current direction. Since the previous request was at 125 and the current is at 143, we'll assume the head is moving upwards (towards higher cylinder numbers).

- **Upward Sweep: 143 -> 913 -> 948 -> 1022 -> 1470 -> 1509 -> 1750 -> 1774 -> 4999 (end of disk)**
- **Downward Sweep: 4999 -> 130 -> 86**
- **Distance: |143-913| + |913-948| + |948-1022| + |1022-1470| + |1470-1509| + |1509-1750| + |1750-1774| + |1774-4999| + |4999-130| + |130-86|**
- **Total Distance: 7084**

4. LOOK

LOOK is similar to SCAN but stops at the last request in each direction, not necessarily the disk ends.

- **Upward Sweep: 143 -> 913 -> 948 -> 1022 -> 1470 -> 1509 -> 1750 -> 1774**
- **Downward Sweep: 1774 -> 130 -> 86**
- **Distance: |143-913| + |913-948| + |948-1022| + |1022-1470| + |1470-1509| + |1509-1750| + |1750-1774| + |1774-130| + |130-86|**
- **Total Distance: 3319**

5. C-LOOK (Circular LOOK)

C-LOOK moves in one direction only, then jumps back to the lowest request.

- **143 -> 913 -> 948 -> 1022 -> 1470 -> 1509 -> 1750 -> 1774 -> 86 -> 130**
- **Distance: |143-913| + |913-948| + |948-1022| + |1022-1470| + |1470-1509| + |1509-1750| + |1750-1774| + |1774-86| + |86-130|**
- **Total Distance: 3371**

Summary Table:

Algorithm	Total Distance
FCFS	6408
SSTF	2363
SCAN	7084
LOOK	3319
C-LOOK	3371

GOOD LUCK