1. Define an operating system.
   An **operating system** is a program that manages the computer hardware. It also provides a basis for application programs and acts as an intermediary between a user of a computer and the computer hardware.
   An **operating system** is software it interfaces between the user and computer.

2. What are the three views of the operating system?
   a. User View
   b. System View
   c. System Goal

3. Define degree of multiprogramming?
   The degree of multiprogramming (m) is the number of jobs existing simultaneously in the system's memory.

4. What is scheduling?
   Scheduling is the activity of deciding which service request execute next on the CPU.

5. What is meant by a batch?
   A batch is a sequence of user job. A computer operator forms a batch by arranging user jobs in a sequence and inserting a special marker cards to indicate the start and end of the batch.
6. What is Multiprogramming?
   The goal of multiprogramming is to exploit the concurrency of operation between the CPU and IO subsystem to achieve high levels of system utilization.

7. What is real time system?
   A real-time system has well-defined, fixed time constraints. Processing *must* be done within the defined constraints, or the system will fail.

8. What is interactive computer system?
   An interactive (or hands-on) computer system provides direct communication between the user and the system.

9. What is meant by Multiprocessing system?
   Systems have more than one processor in close communication, sharing the computer bus, the clock, and sometimes memory and peripheral devices.

10. Name the categories of Multiprocessing system?
    Symmetric multiprocessing
    Asymmetric multiprocessing

11. What is meant by preemption?

    The Forceful deallocation of CPU from one system to another.

12. What is a Distributed system?

    A distributed system is basically a collection of processors interconnected by a communication network in which each processor has its own local memory and other peripherals, and communication between two processors of the system takes places through messages passing over communication network.

13. What are the draw backs of the BP system?

    - The drawbacks of the BP system are CPU utilization is less because whenever any job perform an I/O operation the CPU became idle.
    - In this execution environment, the CPU is often idle, because the speeds of the mechanical I/O devices are intrinsically slower than are those of electronic devices. Even a slow CPU works in the microsecond range, with thousands of instructions executed per second. A fast card reader, on the other hand, might read 1200 cards per minute (or 20 cards per second). Thus, the difference in speed between the CPU and its I/O devices may be three orders of magnitude or more.

14. What is Job?

    Job: Compilation, linking and execution of the program.

15. What is a Process?

    Process: Program in execution.

16. What are the Advantages of the Multiprocessing system?

    a. **Increased throughput**
    b. **Economy of scale**
    c. **Increased reliability**.

17. What is meant CPU bound job?

    CPU bound Job: A program involving a lot of computation and very little IO.

18. What is meant IO bound job?

    IO bund Job: A program involving a lot of IO and very little computations.

19. List any THREE services provide by the Operating System.

    a. Program execution
    b. I /*O* operations
    c. File-system manipulation:

d. Communications:
e. Error detection:
f. Resource allocation:
g. Accounting:
h. Protection

20. List any THREE Components of the Operating System
    a. Process Management:
    b. Main-Memory Management
    c. File Management
    d. I/O-System Management
    e. Secondary-Storage Management
    f. Networking
    g. Protection Systems
    h. Command-Interpreter Systems

1. Explain Time Sharing System.
   Time sharing (or multitasking) is a logical extension of multiprogramming.
   The CPU executes multiple jobs by switching among them, just the switches occur so frequently that the users can interact with each program while it is running.

   An interactive (or hands-on) computer system provides direct communication between the user and the system.

   The user gives instructions to the operating system or to a program directly, using a keyboard or a mouse, and waits for immediate results. Accordingly, the **response time** should be short typically within 1 second or so. .

   A time-shared operating system allows many users to share the computer simultaneously. Since each action or command in a time-shared system tends to be short, only a little CPU time is needed for each user. As the system switches rapidly from one user to the next, each user is given the impression that the entire computer system is dedicated to her use, even though it is being shared among many users.

   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 commonly referred to as a process. When a process executes, it typically executes for only a short time before it either finishes or needs to perform I/O. I/O may be interactive; that is, output is to a display for the user and input is from a user keyboard, mouse, or other device.

Time-sharing operating systems are even more complex than multiprogrammed operating systems. In both, several jobs must be kept simultaneously in memory, so the system must have memory management. To obtain a reasonable response time, jobs may have to be **swapped in** and **out** of main memory to the disk that now serves as a backing store for main memory.

A common method for achieving this goal is **virtual memory**, which is a technique that allows the execution of a job that may not be completely in memory. The main advantage of the virtual-memory scheme is that programs can be larger than physical memory. Further, it abstracts main memory into a large, uniform array of storage, separating logical memory as viewed by the user from physical memory. This arrangement frees programmers from concern over memory-storage limitations. **The time sharing operating system uses the scheduling criteria is Round Robin Scheduling.**

2. What is an Operating System? Explain its services.
   An **operating system** is software it interfaces between the user and computer.

- **Program execution**: The system must be able to load a program into memory and to run that program. The program must be able to end its, execution, either normally or abnormally (indicating error).

- **I /*O* operations**: A running program may require I/O. This I/O may involve a file or an I/O device. For specific devices, special functions may be desired .For efficiency and protection; users usually cannot control I/O devices directly. Therefore, the operating system must provide a means to do I/O.

- **File-system manipulation**: The file system is of particular interest. Obviously, programs need to read and write files. Programs also need to create and delete files by name.

- **Communications**: In many circumstances, one process needs to exchange information with another process. Such communication can occur in two major ways. The first takes place between processes that are executing on the same computer' the second takes place between processes that are executing on different computer systems that are tied together by a computer network. Communications may be implemented via shared memory, or by the technique of message passing, in which packets of information are moved between processes by the operating system.

- **Error detection**: The operating system constantly needs to be aware of possible errors. Errors may occur in the CPU and memory hardware (such as a memory error or a power failure), in I/O devices, (a connection failure on a network, or lack of paper in the printer), and in the user program (such as an arithmetic overflow, an attempt to access an illegal memory location, or a too-great use of CPU time). For each type of error, the operating

system should take the appropriate action to ensure correct and consistent computing.

- **Resource allocation**: When multiple users are logged on the system or multiple jobs are running at the same time, resources must be allocated to each of them. Many different types of resources are managed by the operating system. Some (such as CPU cycles, main memory, and file storage) may have special allocation code, whereas others (such as I/O devices) may have much more general request and release code. For instance, in determining how best to use the CPU, operating systems have CPU-scheduling routines that take into account the speed of the CPU, the jobs that must be executed, the number of registers available, and other factors.

- **Accounting:** We want to keep track of which users use how many and which kinds of computer resources. This record keeping may be used for accounting (so that users can be billed) or simply for accumulating usage statistics.

- **Protection:** The owners of information stored in a multi-user computer system may want to control use of that information. When several disjointed processes execute concurrently, it should not be possible for one process to interfere with the others, or with the operating system itself. Protection involves ensuring that all access to system resources is controlled. Security of the system from outsiders is also important. Such security starts with each user having to authenticate him to the system, usually by means of a password, to be allowed access to the resources. It extends to defending external I/O devices, including modems and network adapters, from invalid access attempts, and to recording all such connections for detection of break-ins. If a system is to be protected and secure, precautions must be instituted throughout it. A chain is only as strong as its weakest link.

3. What are the components of the Operating System? Explain.

1. Process Management:

A process needs certain resources-including CPU time, memory, files, and I/O devices-to accomplish its task. These resources are either given to the process when it is created, or allocated to it while it is running. In addition to the various physical and logical resources that a process obtains when it is created, various initialization data (or input) may be passed along.

We emphasize that a program by itself is not a process; a program is a *passive* entity, such as the contents of a file stored on disk, whereas a process is an *active* entity, with **a program counter specifying the next instruction to execute**. The execution of a process must be sequential. The CPU executes one instruction of the process after another, until the process completes. Further, at any time, at most one instruction is executed on behalf of the process. Thus, although two processes may be associated with the same program, they are nevertheless considered two separate execution sequences. It is common to have a program that spawns many processes as it runs.

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

- **Creating and deleting both user and system processes**
- **Suspending and resuming processes**
- **Providing mechanisms for process synchronization**
- **Providing mechanisms for process communication**
- **Providing mechanisms for deadlock handling.**

2. Main-Memory Management

Main memory is a large array of words or bytes, ranging in size from hundreds of thousands to billions. Each word or byte has its own address. Main memory is a repository of quickly accessible data shared by the CPU and I/O devices. The central processor reads instructions from main memory during the instruction-fetch cycle, and it both reads and writes data from main memory during the data-fetch cycle.

The I/O operations implemented via DMA also read and write data in main memory. The main memory is generally the only large storage device that the CPU is able to address and access directly. For example, for the CPU to process data from disk, those data must first be transferred to main memory by CPU-generated I/O calls. Equivalently, instructions must be in memory for the CPU to execute them.

To improve both the utilization of the CPU and the speed of the computer's response to its users, we must keep several programs in memory. Many different memory-management schemes are available, and the effectiveness of the different algorithms depends on the particular situation. Selection of a memory-management scheme for a specific system depends on many factors -especially on the *hardware* design of the system. Each algorithm requires its own hardware support.

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

- **Keeping track of which parts of memory are currently being used and by whom**
- **Deciding which processes are to be loaded into memory when memory space becomes available**
- **Allocating and deallocating memory space as needed**

3. File Management

File management is one of the most visible components of an operating system. Computers can store information on several different types of physical media. Magnetic tape, magnetic disk, and optical disk are the most common media.

Each of these media has its own characteristics and physical organization. Each medium is controlled by a device, such as a disk drive *or* tape drive, that also has unique characteristics. These properties include access speed, capacity, data-transfer rate, and access method (sequential *or* random).

A file is a collection *of* related information defined by its creator. Commonly, files represent programs (both source and object forms) and data. Data files may be numeric, alphabetic, *or* alphanumeric. Files may be free-form (for example, text files), *or* may be formatted rigidly (for example, fixed fields). A file consists *of* a sequence *of* bits, bytes, lines, *or* records whose meanings are defined by their creators. The concept *of* a file is an extremely general one.

Finally, when multiple users have access to files, we may want to control by whom and in what ways (for example, read, write, append) files may be accessed.

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

- **Creating and deleting files**
- **Creating and deleting directories**
- **Supporting primitives for manipulating files and directories.**
- **Mapping files onto secondary storage**
- **Backing up files on stable (nonvolatile) storage media**

4. I/O-System Management

One of the purposes *of* an operating system is to hide the peculiarities *of* specific hardware devices from the user. The I/O subsystem consists *of*

- **A memory management component that include buffering, caching and spooling.**
- **A general device-driver interface**
- **Drivers for specific hardware devices**

5. Secondary-Storage Management

The main purpose of a computer system is to execute programs. These programs, with the data they access, must be in main memory, or primary storage, during execution. Because main memory is too small to accommodate all data and programs, and because the data that it holds are lost when power is lost, the computer system must provide secondary storage to back up main memory. Most modem computer systems use disks as the principal on-line storage medium, for both programs and data.

Most programs-including compilers, assemblers, sort routines, editors, and formatters-are stored on a disk until loaded into memory, and then use the disk as both the source and destination of their processing. Hence, the proper management of disk storage is of central importance to a computer system.

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

- **Free-space management**
- **Storage allocation.**
- **Disk scheduling**

6. Networking

A distributed system is a collection of processors that do not share memory, peripheral devices, or

a clock. Instead, each processor has its own local memory and clock, and the processors communicate with one another through various communication lines, such as high-speed buses or networks. The processors in a distributed system vary in size and function. They may include small microprocessors, workstations, minicomputers, and large, general-purpose computer systems.

The processors in the system are connected through a communication network, which can be configured in a number of different ways. The network may be fully or partially connected. The communication-network design must consider message routing and connection strategies, and the problems of contention and security.

7. Protection Systems

If a computer system has multiple users and allows the concurrent execution of multiple processes, then the various processes must be protected from one another's activities. For that purpose, mechanisms ensure that the files, memory segments, CPU, and other resources can be operated on by only those processes that have gained proper authorization from the operating system.

For example, memory-addressing hardware ensures that a process '"Can execute only within its own address space. The timer ensures that no process can gain control of the CPU without eventually relinquishing control. Device control registers are not accessible to users, so that the integrity of the various peripheral devices is protected.

Protection is any mechanism for controlling the access of programs, processes, or users to the resources defined by a computer system. This mechanism must provide means for specification of the controls to be imposed and means for enforcement.

Protection can improve reliability by detecting latent errors at the interfaces between component subsystems. Early detection of interface errors can often prevent contamination of a healthy subsystem by another subsystem that is malfunctioning. An unprotected resource cannot defend against use (or misuse) by an unauthorized or incompetent user.

8. Command-Interpreter Systems

One of the most important systems programs for an operating system is the command interpreter, which is the interface between the user and the operating system. Some operating systems include the command interpreter in the kernel. Other operating systems, such as MS-DOS and UNIX, treat the command interpreter as a special program that is running when a job is initiated, or when a user first logs on (on time-sharing systems).

Many commands are given to the operating system by control statements. When a new job is started in a batch system, or when a user logs on to a time-shared system, a program that reads and interprets control statements is executed automatically. This program is sometimes called the control-card interpreter or the command-line interpreter, and is often known as the shell. Its function is simple: To get the next command statement and execute it.

4. What are the THREE view points of the Operating system? Explain.

Operating systems can be explored from three viewpoints**: the user, the system and system goals.**

1 User View

The user view of the computer varies by 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, to maximize the work 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. Performance is important to the user, but it does not matter if most of the system is sitting idle, waiting for the slow I/O speed of the user.

Some users sit at a terminal connected to a mainframe or minicomputer. Other users are accessing the same computer through other terminals. These users share resources and may exchange information. The operating system is designed to maximize resource utilization-to assure that all available CPU time, memory, and I/O are uses efficiently, and that no individual user takes more than her fair share. **Other users sit at workstations**, connected to networks of other workstations and servers. These users have dedicated resources at their disposal, but they also share resources such as networking and servers-file, compute and print servers. Therefore, their operating system is designed to compromise between individual usability and **resource utilization**.

2 System View

We can view an operating system as a resource allocator. A computer system has many resources-hardware and software-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. Facing numerous and possibly conflicting requests for resources, the operating system must decide how to allocate them to specific programs and users so that it can operate the computer system efficiently and fairly.

A slightly different view of an operating system emphasizes the need to control the various I/O devices and user programs. An operating system is a control program. 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.

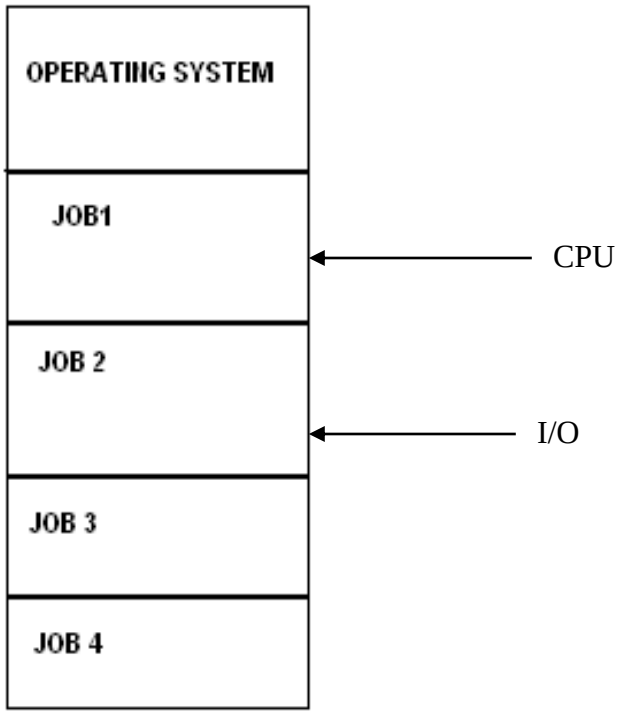
3 System Goals

The primary goal of some operating system **is *convenience for the user.*** Operating systems exist because they are supposed to make it easier to compute with them than without them. This view is particularly clear when we look at operating systems for small PCs.

The primary goal of other operating systems is ***efficient*** **operation** of the computer system. This is the case for large, shared, multi-user systems. These systems are expensive, so it is desirable to make them as efficient as possible.

5. Explain briefly the concept of Multiprogramming.

Multiprogramming increases CPU utilization by organizing jobs so that the CPU always has one to execute. **The goal of multiprogramming is to exploit the concurrency of operation between the CPU and IO subsystem to achieve high levels of system utilization. To optimize the throughput an MP system uses the following concepts.**



1. **A proper Mix of Jobs**: For good throughput it is important to keep both the CPU and IO subsystem busy. Hence the system tries to keep an appropriate number of CPU bound and IO bound jobs in execution.
   a). **CPU bound Job**: A program involving a lot of computation and very little IO.
   b).**IO bund Job**: A program involving a lot of IO and very little computations.
2. **Preemptive and priority based scheduling**: Scheduling is priority based ie, the CPU is always allocated to the highest priority job which wishes use it. Thus a low priority job executing on the CPU is preempted if a higher priority job wishes to use the CPU.
3. **Degree of multiprogramming:** The degree of multiprogramming (m) is the number of jobs existing simultaneously in the system's memory.

The idea is as follows: The operating system keeps several jobs in memory simultaneously (Figure). This set of jobs is a subset of the jobs kept in a job pool-since the number of jobs that can be kept simultaneously in memory is usually much smaller than the number of jobs that can be in the job pool. The operating system picks and begins to execute one of the jobs in the memory. Eventually, the job

may have to wait for some task, such as an I/O operation, to complete. In a non-multiprogrammed system, the CPU would sit idle. In a multiprogramming system, the operating system simply switches to, and executes, another job. When *that* job needs to wait, the CPU is switched to *another* job, and so on. Eventually, the first job finishes waiting and gets the CPU back. As long as at least one job needs to execute, the CPU is never idle.

6. Write a note Real Time System

   Another form of a special-purpose operating system is the real-time system. A real-time system is used when rigid time requirements have been placed on the operation of a processor or the flow of data; thus, it is often used as a control device in a dedicated application. Sensors bring data to the computer.

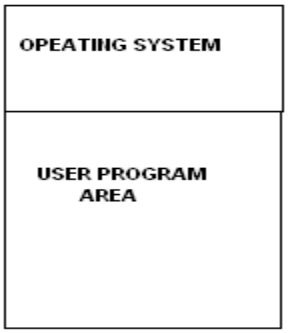
   The computer must analyze the data and possibly adjust controls to modify the sensor inputs. Systems that control scientific experiments, medical imaging systems, Satellite communication, industrial control systems, and certain display systems are real-time systems. Some automobile-engine fuel-injection systems, home-appliance controllers, and weapon systems are also real-time systems.

   A real-time system has well-defined, fixed time constraints. Processing *must* be done within the defined constraints, or the system will fail. A real-time system functions correctly only if it returns the correct result within-its time constraints. Contrast this requirement to a time-sharing system, where it is desirable (but not mandatory) to respond quickly, or to a batch system, which may have no time constraints at all.

   The Real Time system uses the scheduling criteria is **deadline oriented scheduling.**

7. Explain briefly the concept of Batch Processing System.

What do Early computers were physically enormous machines run from a console. The common input devices were card readers. The common output devices were line printers. The user did not interact directly with the computer systems. Rather, the user prepared a job -which consisted of the program, the data, and control information about the nature of the job (control cards) submitted it to the computer operator. The job was usually in the form of punch cards. At some later time (after minutes, hours, or days), the output appeared. The output consisted of the result of the program, as well as a dump of the final memory and register contents for debugging.



The operating system in these early computers was fairly simple. Its major task was to transfer control automatically from one job to the next. The operating system was always resident in memory.

A batch is a sequence of user job. A computer operator forms a batch by arranging user jobs in a sequence and inserting a special marker cards to indicate the start and end of the batch. After forming a batch the operator submits it to the batch processing operating system. The primary function of BP system is to implement the processing of the jobs in a batch without requiring any interventions of the operator. This achieved by automating the transition from the execution of one job to that of the next job in the batch. Results of a batch released to the user at the end of the batch.

To speed up processing, operators batched together jobs with similar needs and ran them through the computer as a group. Thus, the programmers would leave their programs with the operator. The operator would sort programs into batches with similar requirements and, as the computer became available, would run each batch. The output from each job would be sent back to the appropriate programmer.

The BP system used the scheduling criteria is **First Come First Serve (FCFS)**

- **The drawbacks of the BP system are CPU utilization is less because whenever any job perform an I/O operation the CPU became idle**.
- **In this execution environment, the CPU is often idle, because the speeds of the mechanical I/O devices are intrinsically slower than are those of electronic devices. Even a slow CPU works in the microsecond range, with thousands of instructions executed per second. A fast card reader, on the other hand, might read 1200 cards per minute (or 20 cards per second). Thus, the difference in speed between the CPU and its I/O devices may be three orders of magnitude or more.**

8. What do you mean by SPOOLING?

   Spooling: (**Simultaneous peripheral operation on-line**) A Spool is a buffer that holds output for a device, such as printer, that cannot accept interleaved data streams. Although a printer can serve only one job at a time, several applications may wish to print their output concurrently, without having their output mixed together. The operating system solves this problem by intercepting all output to the printer. Each applications output is spooled to a separate disk file. When an application finishes printing, the spooling system copies the queued spool file to the printer one at a time.

9. Explain Multiprocessing system?

Most systems to date are single-processor systems; that is, they have only one main CPU. However, multiprocessor systems (also known as parallel systems or tightly coupled systems) are growing in importance**. Such systems have more than one processor in close communication, sharing the computer bus, the clock, and sometimes memory and peripheral devices.**

Multiprocessor systems have three main advantages.

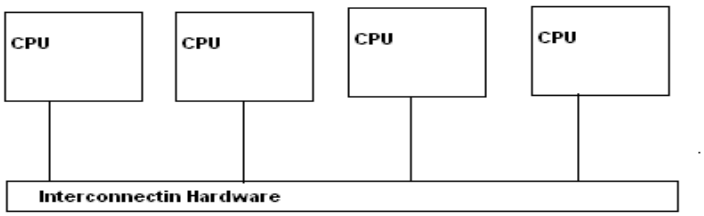
1. Increased throughput. By increasing the number of processors, we hope to get more work done in less time. The speed-up ratio with N processors is not N; rather, it is less than N. When multiple processors cooperate on a task, a certain amount of overhead is incurred in keeping all the parts working correctly.

2**. Economy of scale**. Multiprocessor systems can save more money than multiple single-processor systems, because they can share peripherals, mass storage, and power supplies. If several programs operate on the same set of data, it is cheaper to store those data on one disk and to have all the processors share them, than to have many computers with local disks and many copies of the data.

3. Increased reliability. If functions can be distributed properly among several processors, then the failure of one processor will not halt the system, only slow it down. If we have ten processors and one fails, then each of the remaining nine processors must pick up a share of the work of the failed processor. Thus, the entire system runs only 10 percent slower, rather than failing altogether. This ability to continue providing service proportional. *to* the level of surviving hardware is called **graceful degradation**. Systems designed for **graceful degradation** are also called **fault tolerant**.

The *most* common multiple-processor systems no use **symmetric multiprocessing (SMP**), in which each processor runs an identical *copy* of the operating system, and these copies communicate with one another as needed.

Some systems use **asymmetric multiprocessing** which each processor is assigned a specific task. A master processor controls the system; the other processors either look *to* the master for instruction or have predefined tasks. This scheme defines a master-slave relationship. The master processor schedules and allocates work *to* the slave processors.



One Marks Questions

1. Define the term process.

Informally, a process is a program in execution.

A process is a tuple

(Process id, code, data, registers values, pc values).

Where process id is unique in the system,

Code is the program code

Data is the data used during its execution

Register values are the values in the machine registers and

Pc Values is the address in the program counter

2. What is job scheduling?

Job scheduling: is process of selecting the job from job queue and loads them into memory for execution

3. What is process scheduling?

Process Scheduling: is the process of selects the jobs that are ready to execute and allocates the CPU to one of them.

4. What is a long term scheduling?

The long-term scheduler, or **job scheduler,** selects processes from this pool and loads them into memory for execution.

5. What is a Short term scheduling?

The **short-term scheduler, or CPU scheduler**, selects from among the processes that are ready to execute, and allocates the CPU to one of them.

6. What is medium term scheduling?

Removes processes from memory and thus reduces the degree of multiprogramming. At some later time, the process can be reintroduced into memory and its execution can be continued where it left off. This scheme is **called swapping**. The process is **swapped out, and is later swapped in**, by the medium-term scheduler.

7. State True or False

The process waiting to be assigned to a processor is said to be in waiting state.

False

8. What is thread cancellation?

Thread Cancellation is the task of terminating a thread before it has completed.

9. What is meant by Context Switch?

Switching the CPU to another process requires saving the state of the old process and loading the saved state for the new process. **This task is known as a context switch.**

10. What do you mean by Cooperating process?

The concurrent processes executing in the operating system may be either **independent processes or cooperating processes**.

a process is cooperating if it can affect or be affected by the other processes executing in the system

11. Define the term PCB.

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

A data structure called the process control block (PCB) is used by an operating system to keep track of all necessary information concerning a process.

12. What is meant by System Calls?

System calls provide the interface between a process and the operating system.

13. What is a thread?

A thread is subdivision of a process. A thread, sometimes called a lightweight process (LWP), is a basic unit of CPU utilization; it comprises a thread id, a program counter, a register set, and a stack. It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals.

14. What is a Scheduler?

A process migrates between the various scheduling queues throughout its lifetime. The operating system must select, for scheduling purposes, processes from these queues in some fashion. The selection process is carried out by the appropriate scheduler.

15. What is a User threads?

User threads are supported above the kernel and are implemented by a thread library at the user level.

16. What is a System threads?

Kernel threads are supported directly by the operating system

17. What is a Job Queue?

As processes enter the system, they are put into a **job queue**. This queue consists of all processes in the system.

18. What is Ready Queue?

The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the **ready queue**.

19. What is Device Queue?

The list of processes waiting for a particular I/O device is called a **device queue.**

20. State True or False.

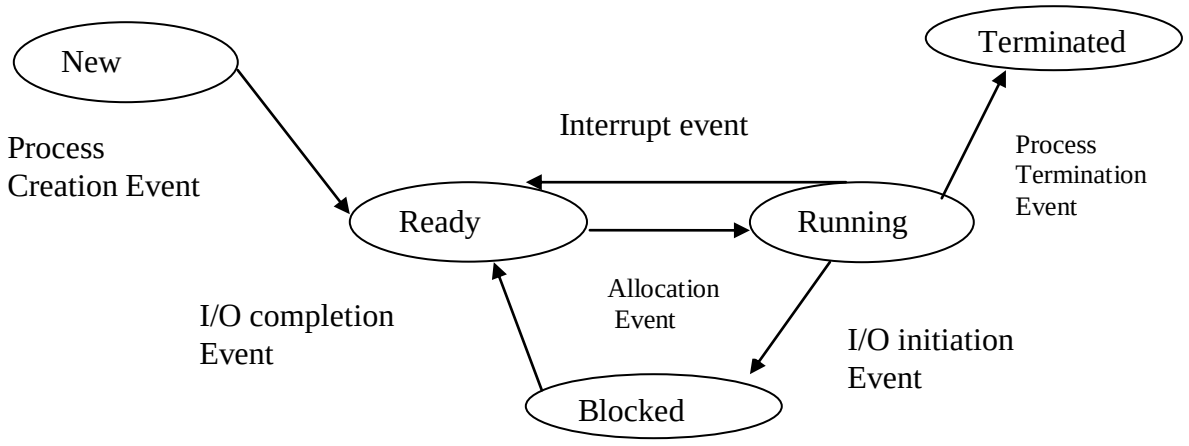
A CPU bound process is one generate I/O requests frequently.

False

Five Marks Questions

1.Explain Process State transition with a neat diagram.

A state transition is a change from one state to another. A state transition is caused by the occurrence of some events in the system. When a process in the running state makes an I/O request, it has enters the blocked state awaiting completions of an I/O. When I/O completes the process state changes from blocked to ready state



Process State Transition Diagram

Events pertaining to a process

1. Request event: Process makes a resource request
2. Allocation event: A requested resource is allocated.
3. I/O initiation event: Process wishes to start I/O.
4. I/O completion event: An I/O operation completes.
5. Timer interrupt: A system timer indicates end of a time interval.
6. Process creation event: A new process is created.
7. Process termination event: A process finishes its executions.

2. What is Scheduler? Explain different types of scheduler.

A process migrates between the various scheduling queues throughout its lifetime. The operating system must select, for scheduling purposes, processes from these queues in some fashion. The selection process is carried out by the appropriate scheduler.

long-term scheduler, or **job scheduler:** selects processes from this pool and loads them into memory for execution.

Short-term scheduler, or CPU scheduler: selects from among the processes that are ready to execute, and allocates the CPU to one of them.

Medium-term scheduler: removes processes from memory (and from active contention for the CPU), and thus reduces the degree of multiprogramming. At some later time, the process can be reintroduced into memory and its execution can be continued where it left off. This scheme is **called swapping**. The process is **swapped out, and is later swapped in**, by the medium-term scheduler.

3.What is a PCB? Explain the different components of the PCB with a diagram.

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

A data structure called the process control block (PCB) is used by an operating system to keep track of all necessary information concerning a process.

- **Process state**: The state may be new, ready, running, waiting, halted, and soon.
- **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. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward
- **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.
- ***I /O* status information**: The information includes the list of I/O devices allocated to this process, a list of open files, and so on.

Process Id
Priority
Process State
Program Counter
Registers
Memory Allocation
Event Information
PCB Pointer

Process Control Block

4. Explain the User and Kernel threads.

User threads are supported above the kernel and are implemented by a thread library at the user level. The library provides support for thread creation, scheduling, and management with no support from the kernel. Because the kernel is unaware of user-level threads, all thread creation and scheduling are done in user space without the need for kernel intervention. Therefore, user-level threads are generally fast to create and manage; they have drawbacks, however. For instance, if the kernel is single-threaded, then any user-level thread performing a blocking system call will cause the entire process to block, even if other threads are available to run within the application.

Kernel threads are supported directly by the operating system: The kernel performs thread creation, scheduling, and management in kernel space. Because thread management is done by the operating system, kernel threads are generally slower to create and manage than are user threads. However, since the kernel is managing the threads, if a thread performs a blocking system call, the kernel can schedule another thread in the application for execution.

5.What are the benefits of the Multithread programming?

The benefits of multithreaded programming can be broken down into four major categories:

1. Responsiveness: Multithreading an interactive application may allow a program to continue running even if part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to the user. For instance, a multithreaded web browser could still allow user interaction in one thread while an image is being loaded in another thread.

2. Resource sharing: By default, threads share the memory and the resources of the process to which they belong. The benefit of code sharing is that it allows an application to have several different threads of activity all within the same address space.

3. Economy: Allocating memory and resources for process creation is costly. Alternatively, because threads share resources of the process to which they belong, it is more economical to create and context switch threads.

4. Utilization of multiprocessor architectures: The benefits of multithreading can be greatly increased in a multiprocessor architecture, where each thread may be running in parallel on a different processor. A single-threaded process can only run on one CPU, no matter how many are available. Multithreading on a multi-CPU machine increases concurrency. In single processor architecture, the CPU generally moves between each thread so quickly as to create an illusion of parallelism, but in reality only one thread is running at a time.

7. Explain the Advantages of Cooperating Processes.

Advantages of Cooperating Processes

- **Information sharing**: Since several users may be interested in the same piece of information (for instance, a shared file), we must provide an environment to allow concurrent access to these types of resources. .
- **Computation speedup**: If we want a particular task to run faster, we must break it into subtasks, each of which will be executing in parallel with the others. Such a speedup can be achieved only if the computer has multiple processing elements (such as CPUs or I/O channels).
- **Modularity:** We may want to construct the system in a modular fashion, dividing the system functions
- **Convenience:** Even an individual user may have many tasks on which to work at one time. For instance, a user may be editing, printing, and compiling in parallel.

8. Write a note on threads.

A thread is subdivision of a process. A thread, sometimes called a lightweight process (LWP), is a basic unit of CPU utilization; it comprises a thread id, a program counter, a register set, and a stack. It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals.

8. Explain process scheduling using Queuing diagram.

As processes enter the system, they are put into a **job queue**.

This queue consists of all processes in the system. The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the **ready queue**.

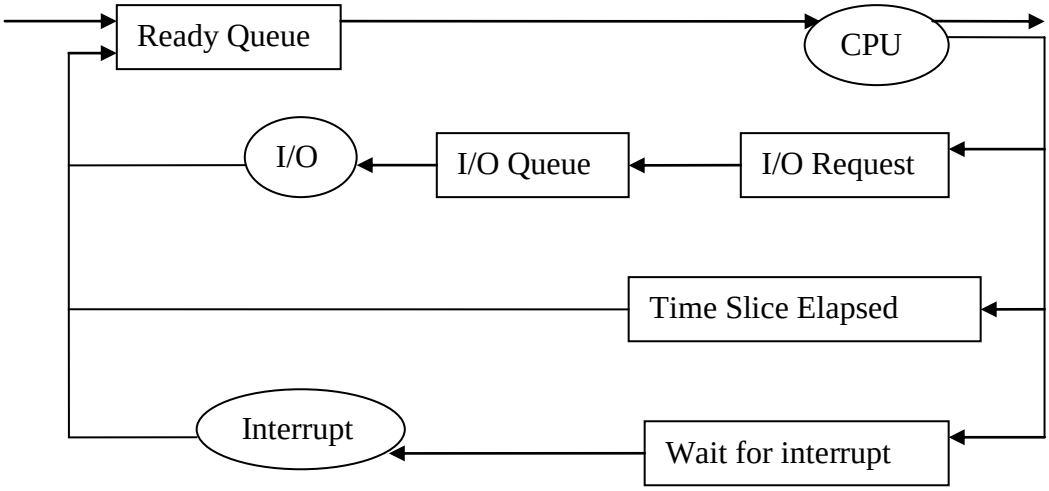
This queue is generally stored as a linked list. A ready-queue header contains pointers to the first and final PCBs in the list. We extend each PCB to include a pointer field that points to the next PCB in the ready queue.

The operating system also has other queues. When a process is allocated the CPU, it executes for a while and eventually quits, is interrupted, or waits for the occurrence of a particular event, such as the completion of an I/O request. In the case of an I/O request, such a request may be to a dedicated tape drive, or to a shared device, such as a disk. Since the system has many processes, the disk may be busy with the I/O request of some other process. The process therefore may have to wait for the disk. The list of processes waiting for a particular I/O device is called a **device queue**. Each device has its own device queue.

A common representation of process scheduling is a queuing diagram, such as that in Figure Each **rectangular box** represents a queue. Two types of queues are present: **the ready queue** and a set of **device queues**. **The circles'** represent the resources that serve the queues, and the arrows indicate the flow of processes in the system. A new process is initially put in the ready queue. It waits in the ready queue until it is selected for execution (or dispatched). Once the process is assigned to the CPU and is executing, one of several events could occur:

- **The process could issue an I/O request, and then be placed in an I/O queue.**
- **The process is preempted could be removed forcibly from the CPU, and be put back in the ready queue.**
- **The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.**

In the first two cases, the process eventually switches from the waiting state to the ready state, and is then put back in the ready queue. A process continues this cycle until it terminates, at which time it is removed from all queues and has its PCB and resources deallocated.



Queuing Diagram representation of process Scheduling

10.Write a note on process.

Informally, a process is a program in execution.
A process is a tuple

(Process id, code, data, registers values, pc values).

Where process id is unique in the system,

Code is the program code

Data is the data used during its execution

Register values are the values in the machine registers and

Pc Values is the address in the program counter

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**. In addition, a process generally includes the process stack, which contains temporary **data (**such as method parameters, return addresses, and local variables), and a data section, which contains global variables.

We emphasize that a program by itself is not a process; a program is a ***passive*** **entity,** such as the contents of a file stored on disk, whereas a process is an ***active*** **entity,** with a program counter specifying the next instruction to execute and a set of associated resources.

1. What is meant by CPU Utilization?

It means how much time CPU is busy in the system.

2. Define the term Aging.

Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time.

3. Define Turnaround Time.

The interval from the time of submission of a process to the time of completion is the turnaround time.

4. State True or False.

Turnaround Time is the sum of the periods spends wait in the ready queue.

False

5. Define Response Time.

Response is the time from the submission of a request until the first response is produced.

6. What is Throughput?

The number of process executed per Unit time. The rate may be 1 process per unit or 10 processes per unit time

7. Define Preemption.

Preemption is the process of forceful de-allocation of the CPU from the process.

8. Define Priority.

Priority is a notation used in a scheduler to decide which process should be scheduled when many process await service

9. Define Time-Slice.

The time slice is the largest amount of CPU time any process can consume when scheduled execute on the CPU.

10. What is Waiting Time?

Waiting time is the sum of the periods spent waiting in the ready queue.

11. What is Non-Preemptive Scheduling?

Under Non-Preemptive Scheduling once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state

12. What is Preemptive Scheduling?

Preemptive Scheduling can be switched to the processing of a new request before completing the processing of a request scheduled earlier. In Preemptive scheduling the preemption is occurred.

13. The number of process executed per time unit is called……Throughput.…….

14. Define Arrival time.

It is the time at which process is arrived in to the system.

15. Define Deadline of a job.

It is the maximum time at which system expects the output of the process.

16. What are the disadvantages of FCFS?

Disadvantages of FCFS

- **Average waiting times is more.**
- **Starvation of the short process.**

17. What are the disadvantages of SJF?

Disadvantages of SJF

- The real difficulty with the SJF algorithm estimation of the length of the next CPU request.
- Starvation of the long Process

Five Marks Questions

1. Explain Priority CPU Scheduling with an example.

A priority is associated with each process, and the CPU is allocated to the process with the highest priority. Equal-priority processes are scheduled in FCFS order.

An SJF algorithm is simply a priority algorithm where the priority (p) is the inverse of the (predicted) next CPU burst. The larger the CPU burst, the lower the priority, and vice versa.

Note that we discuss scheduling in terms of *high* priority and *low* priority. Priorities are generally some fixed range of numbers, such as 0 to 7, or 0 to 4,095. However, there is no general agreement on whether 0 is the highest or lowest priority. Some systems use low numbers to represent low priority; others use low numbers for high priority.

As an example, consider the following set of processes, assumed to have arrived at time 0, in the order P1, *P2, P3, P4,* P5, with the length of the CPU-burst time given in milliseconds:

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

Using priority scheduling, we would schedule these processes according to the following Gantt chart:

P2	P5	P1	P3	P4

The average waiting time is 8.2 milliseconds.
Thus the average Turnaround time is=(16+1+18+19+6)=12 milliseconds

2. Explain FCFS CPU Scheduling with an example.

First-Come First-Served (FCFS) scheduling :

With this scheme, the process that requests the CPU first is allocated the CPU first. The implementation of the FCPS policy is easily managed with a FIFO queue. When a process enters the ready queue, its PCB is linked onto the tail of the queue. When the CPU is free, it is allocated to the process at head of the queue. The running process is then removed from the queue.

The average waiting time under the FCPS policy, however, is often quite long. Consider the following set of processes that arrive at time 0 with the length of the CPU-burst time given in milliseconds:

Process	Burst Time
P1	24
P2	3
P3	3

If the processes arrive in the order P1, *P2, P3,* and are served in FCPS order, we get the result shown in the following **Gantt chart.**

P1	P2	P3

The waiting time is 0 milliseconds for process P1, 24 milliseconds for process *P2,* and 27 milliseconds for process *P3*

Thus the average waiting time is (0 + 24 + 27)/3 = 17 milliseconds.
Thus the average Turnaround time is (24+27+30)/3=27 milliseconds.

If the processes arrive in the order *P2, P3,* P1, however, the results will be as shown in the following

Gantt chart:

P2	P3	P1

Thus the average waiting time is (6 + 0 + 3)/3 = 3 milliseconds.
Thus the average Turnaround time is (30+3+6)/3=35 milliseconds.

Thus the average waiting time under a FCFS policy is generally not minimal, and may vary substantially if the process CPU-burst times vary greatly.

3. Explain the different scheduling criteria for CPU scheduling algorithms.

Scheduling Criteria

1. CPU utilization: CPU utilization is measured in terms of percentage. It means how much time CPU is busy in the system. CPU utilization may ranges from 0 to 100 percent.

2. Throughput (H): The number of process executed per Unit time. The rate may be 1 process per unit or 10 processes per unit time.

3. Turnaround time (ta): **The interval from the time of submission of a process to the time of completion is the turnaround time.** Turnaround time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O.

4. Waiting time: The CPU-scheduling algorithm does not affect the amount of time during which a process executes or does I/O; it affects only the amount of time that a process spends waiting in the ready queue. **Waiting time is the sum of the periods spent waiting in the ready queue.**

5. Response time: In an interactive system, turnaround time may not be the best criterion. Often, a process can produce some output fairly early, and can continue computing new results while previous results are being output to the user. **Thus, another measure is the time from the submission of a request until the first response is produced**.

6. Time Slice: **The time slice is the largest amount of CPU time any process can consume when scheduled execute on the CPU.**

7. N: No of Processes

8. CPU Burst: Total CPU time required for the execution of a process.

9. Arrival Time: It is the time at which process is arrived in to the system.

10. Deadline: It is the maximum time at which system expects the output of the process.

11. Priority: Priority is a notation used in a scheduler to decide which process should be scheduled when many process await service

4. Explain SJF CPU Scheduling with an example.

Shortest-Job-First Scheduling (SJF):

When the CPU is available, it is assigned to the process that has the smallest next CPU burst. **The SJF scheduler always schedules the shortest of arrived process.** If two processes have the same length next CPU burst, FCFS scheduling is used to break the tie. Note that a more appropriate term would be the *shortest next CPU burst,* because the scheduling is done by examining the length of the next CPU burst of a process, rather than its total length.

As an example, consider the following set of processes, with the length of the CPU-burst time given in milliseconds.

Process	Burst Time
P1	6
P2	8
P3	7
P4	3

Using SJF scheduling, we would schedule these processes according to the following Gantt chart:

P4	P1	P3	P2

The waiting time is 3 milliseconds for process Pl, 16 milliseconds for process *P2,* 9 milliseconds for process *P3,* and 0 milliseconds for process *P4.*

Thus the average waiting time is (3 + 16 + 9 + 0) /4 = 7 milliseconds.

Thus the average Turnaround time is= (9+24+16+3)/4=13 milliseconds.

The SJF scheduling algorithm is provably *optimal,* in that it gives the minimum average waiting time for a given set of processes. By moving a short process before a long one, the waiting time of the short process decreases more than it increases the waiting time of the long process. Consequently, the *average* waiting time decreases.

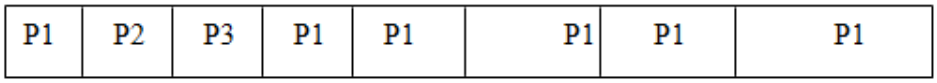
5. Explain RR CPU Scheduling with an example.

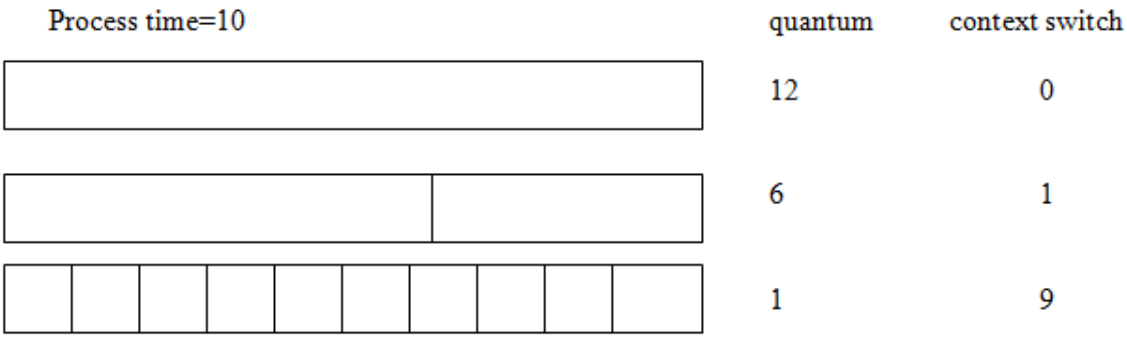
Round-Robin Scheduling:

The round-robin (RR) scheduling algorithm is designed especially for timesharing systems. It is similar to FCFS scheduling, but preemption is added to switch between processes. A small unit of time, called a time quantum (or time slice), is defined. A time quantum is generally from 10 to 100 milliseconds. The ready queue is treated as a circular queue. The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum.

Process	Burst Time
P1	24
P2	3
P3	3

If we use a time quantum of 4 milliseconds, then process P1 gets the first 4 milliseconds. Since it requires another 20 milliseconds, it is preempted after the first time quantum, and the CPU is given to the next process in the queue, process *P2*. Since process P2 does not need 4 milliseconds, it quits before its time, quantum expires. The CPU is then given to the next process, process *P3*. Once each process has received 1 time quantum, the CPU is returned to process P1 for an additional time quantum. The resulting RR schedule is





In the RR scheduling algorithm, no process is allocated the CPU for more than 1 time quantum in a row. If a process CPU burst exceeds 1 time quantum, that process is *preempted* and is put back in the ready queue. The RR scheduling algorithm is preemptive.

If there are *n* processes in the ready queue and the time quantum is *q*, then each process gets *1/n* of the CPU time in chunks of at most *q* time units. Each process must wait no longer than *(n* - 1) x *q* time units until its next time quantum. For example, if there are five processes, with a time quantum of 20 milliseconds, then each process will get up to 20 milliseconds every 100 milliseconds.

6. Explain Preemptive and Non- Preemptive Scheduling.

Under Non-Preemptive Scheduling once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.

Preemptive Scheduling can be switched to the processing of a new request before completing the processing of a request scheduled earlier. In Preemptive scheduling the preemption is occurred.

1. What is Race condition?

A race condition is arises due to the execution operation a_i and a_j on data ds if the value of ds after completing the execution of ai and aj is neither $f_i(f_j(ds))$ nor $f_j(f_i(ds))$

2. Define the term Semaphore.

A semaphore S is an integer variable that, apart from initialization, is accessed only through two standard atomic operations wait and signal.

3. What is Binary semaphore?

A binary semaphore is a semaphore with an integer value that can range only between 0 and 1

4. What is a Mutual Exclusion?

If process Pi is executing in its critical section, then no other processes can be executing in their critical sections. (At most only one process can execute in the critical section at any time)

5. What is a Critical Section?

The sections of a program that need exclusive access to a shared resource are referred as critical section.

6. What is progress condition?

If no process is executing in its critical section and some processes wish to enter their critical sections will be granted entry to the CS.

7. What is Bounded-Wait condition?

There exists a bound on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

8. Define Interacting process.

Interacting Process: Process Pi and Pj are interacting Process if

($read\_set_i$ U $Write\_set_i$) ($read\_set_j$ U $Write\_set_j$)

Where **$read\_set_i$** **:** Set of data item read by process Pi

$Write\_set_i$: Set of data item write by process Pi

Five Marks Questions

1. Write a note on a Critical section.

Consider a system consisting of *n* processes {Po, P1, ..., Pn}. Each process has a segment of code, called a critical section, in which the process may be changing common variables, updating a table, writing a file, and so on. **Important feature of the system is that, when one process is executing in its critical section, no other process is to be allowed to execute in its critical section. Thus, the execution of critical sections by the processes is *mutually exclusive* in time**. The *critical-section* problem is to design a protocol that the processes can use to cooperate. Each process must request permission to enter its critical section. The section of code implementing this request is **the entry section**. The **critical section** may be followed by an **exit section**. The remaining code is the **remainder section.**

Critical Section: **The sections of a program that need exclusive access to a shared resource are referred as critical section.**

OR

A critical section for a data item d is a section of code which can not be executed concurrently with itself or with other critical section for d.

Do {

Entry section

Critical section

Exit section

Remainder section

} while (1);

Figure **General Structure of a typical process Pi.**

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

1. Mutual Exclusion (Correctness): If process Pi is executing in its critical section, then no other processes can be executing in their critical sections. (At most only one process can execute in the critical section at any time)

2. Progress: If no process is executing in its critical section and some processes wish to enter their critical sections will be granted entry to the CS.

3. Bounded Waiting: There exists a bound on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted

2. What is Semaphore? Explain its usage and implementation.

A semaphore S is an integer variable that, apart from initialization, is accessed only through two standard atomic operations wait and signal.

We can use semaphores to deal with the n-process critical-section problem. The *n* processes share a semaphore, mutex (standing for **mutual exclusion***),* initialized to 1. Each process Pi is organized as shown in Figure.

We can also use semaphores to solve various synchronization problems. For example, consider two concurrently running processes: P1 with a statement S1 and P2 with a statement S2. Suppose that we require that S2 be executed only after S1 has completed. We can implement this scheme readily by letting PI and P2 share a common semaphore mutex, initialized to 0, and by inserting the statements

S1;
Signal (mutex);

in process p1, and the statements

Wait (mutex);
S2;

in process *P2.* Because synch is initialized to 0, P2 will execute S2 only after P1 has invoked signal (mutex) which is after S1

```
Do
 {
      Wait (mutex);

      Critical section
      Signal (mutex);

      Remainder section
} While (1);
```

Mutual-exclusion implementation with semaphores.

3. What are the requirements of the Critical section algorithms?

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

1. Mutual Exclusion (Correctness): If process Pi is executing in its critical section, then no other processes can be executing in their critical sections. (At most only one process can execute in the critical section at any time)

2. Progress: If no process is executing in its critical section and some processes wish to enter their critical sections will be granted entry to the CS.

3. Bounded Waiting: There exists a bound on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that

request is granted.

Define the term Deadlock.

Deadlock: A deadlock involving a set of process Pi in D is a situation in which

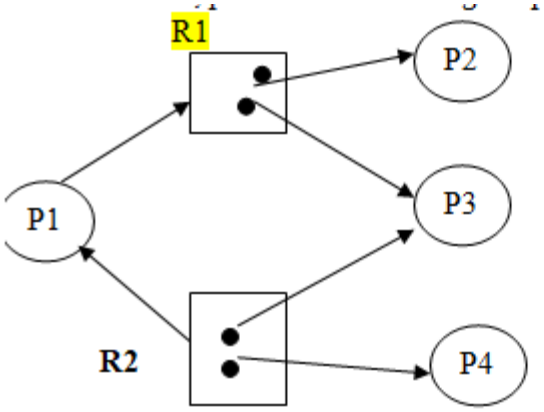
1. Every process *Pi in D blocked on some event ei.*

2. Every event ei can only caused by some process in D.

Or

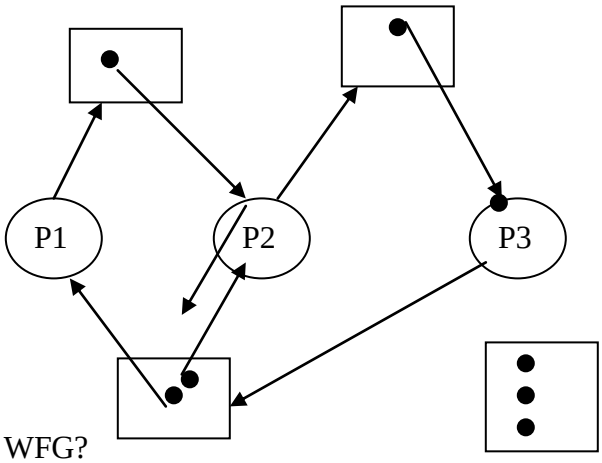
Deadlock is situation in which a set process unknowingly waiting for the resources which is not available

1. Draw a Resource Allocation Graph with a Cycle but no Deadlock.



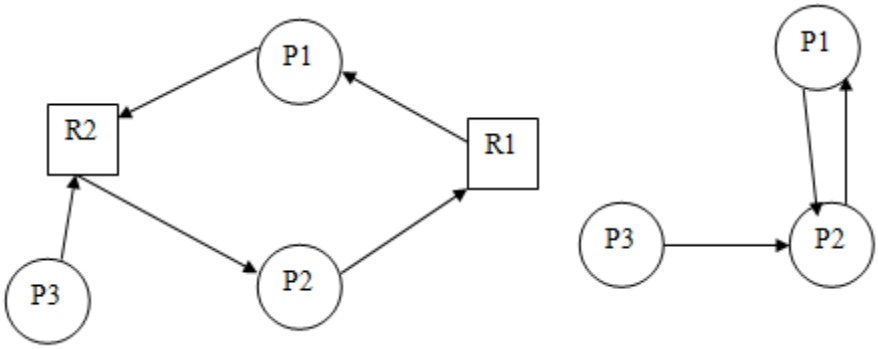
Resource-allocation graph with a cycle but no deadlock.

2. Draw a Resource Allocation Graph with a Cycle and Deadlock.



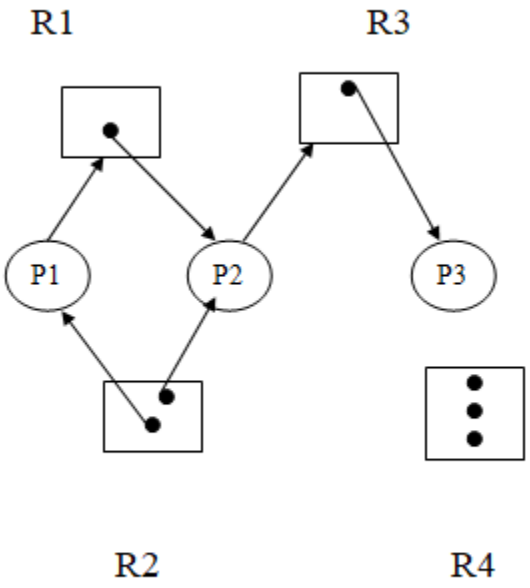
3. What is a WFG?

When all the resource types have only a single unit each, a simplified from of resource allocation graph is normally used. The simplified graph is obtained from the original resource allocation graph by removing the resource nodes and collapsing the appropriate edges.



4. What is a RAG?

Deadlocks can be described more precisely in terms of a directed graph called a system resource-allocation graph. This graph consists of a set of vertices V and a set of edges E. The set of vertices V is partitioned into two different types of nodes P = {PI, P2, ..., Pn}, the set consisting of all the active processes in the system, and R = {Rl, R2, ..., Rm}, the set consisting of all resource types in the system.



Resource-allocation graph

5. What Request edge is signifies in a Resource Allocation Graph.

When process Pi *requests* an instance of resource type Rj. a *request* edge is inserted in the resource-allocation graph.

A directed edge from process Pi to resource type Rj is denoted by Pi Rj; it signifies that →

process Pi requested an instance of resource type Rj and is currently waiting for that resource.

6. What Assignment edge is signifies in a Resource Allocation Graph.

When this *request* can be fulfilled, the *request* edge is *instantaneously* transformed to an assignment edge.

. A directed edge from resource type Rj to process Pi is denoted by Rj → Pi; it signifies that an instance of resource type Rj has been allocated to process Pi. A directed edge Pi → Rj is called a **request edge;** a directed *edge Rj* → *Pi* is called an **assignment edge**.

7. What is meant by a safe request?

A request is safe request at time T if after granting a request at time T there exists at least one sequence of allocation and deal location all the process in the system can complete

8. State True or False.

In a Resource Allocation Graph, if each resource type has exactly one instance, then a cycle implies that a Deadlock has occurred.

True

9. What is a Reachable set?

The reachable set of a node A is the set of all nodes B such that a path exists from A to B.

10. What is a Knot?

A knot is a nonempty set K of nodes such that the reachable set of each node in K is exactly the set K. A knot always contain one or more cycle.

11. What is meant by Deadlock Prevention.?

Deadlock prevention is a set of methods for ensuring that at least one of the necessary conditions cannot hold. These methods prevent deadlocks by constraining how requests for resources can be made.

12. What is meant by Deadlock Avoidance?

Deadlock avoidance, on the other hand, requires that the operating system be given in advance additional information concerning which resources a process will request and use during its lifetime.

Five Marks Questions

1. Explain Bankers Algorithm with an example.

Banker's Algorithm

The resource-allocation graph algorithm is not applicable to a resource allocation system with multiple instances of each resource type. The deadlock-avoidance algorithm that we describe next is applicable to such a system, but is less efficient than the resource-allocation graph scheme. This algorithm is commonly known as the *banker's algorithm.* The name was chosen because this algorithm could be used in a banking system to ensure that the bank never allocates its available cash such that it can no longer satisfy the needs of all customers.

When a *new* process enters the system, it must declare the maximum number of instances of each resource type that it may need. This number may not exceed the total number of resources in the system. When a user requests a set of resources, the system must determine whether the allocation of these resources will leave the system in a safe state. If it will, the resources are allocated; otherwise, the process must wait until some other process releases enough resources.

Let *n* be the number of processes in the system and *m* be the number of resource types. We need the following data structures:

Available: A vector of length *m* indicates the number of available resources of each type. If *Available[i,j]* = k there are *k* instances of resource type Rj available.

. Max: An *n x m* matrix defines the maximum demand of each process. If *Max [i,j] = k,* then process Pi; may request at most *k* instances of resource type *Rj*.

Allocation: An *n* x *m* matrix defines the number of resources of each type currently allocated to each process. If *Allocation [i,j] = k,* then process Pi is currently allocated *k* instances of resource type *Rj*

. Need: An *n* x *m* matrix indicates the remaining resource need of each process. If *Need [i, j] = k,* then process Pi may need *k* more instances of resource type Rj to complete its task.

Note that ***Need [i,j] = Max[i,j] +Allocation [i,j].***

1. Safety Algorithm

The algorithm for finding out whether or not a system is in a safe state can be described as follows:

1. Let *Work* and *Finish* be vectors of length *m* and *n,* respectively.

Initialize *Work: = Available* and *Finish[i]:= false* for i = 1 to n.

2. Find an i such that both
   a. *Finish[i] = false*
   b. *Need <= Work.*
   If no such *i* exists, go to step 4.

3. *Work: = Work + Allocation*
Finish[i]:= true
   Go to step 2.

4. If *Finish[i] = true* for all i, then the system is in a safe state.

This algorithm may require an order of $m \times n^2$ operations to decide whether a state is safe.

2 Resource-Request Algorithms

Let $Request_i$ be the request vector for process *Pi*. If $Request_i[j] = k$, then process Pj wants *k* instances of resource type *Rj*. When a request for resources is made by process *Pi* the following actions are taken:

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

2. If *Request i*<=; *Available*, go to step 3. Otherwise, Pj must wait, since the resources are not available.

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

Available: = Available – Request $_i$ *;*
Allocation $_i$ *:= Allocation* $_i$ *+ Request* $_i$ *;*
Need i: = Need i – Request i;

If the resulting resource-allocation state is safe, the transaction is completed and process Pi is allocated its resources. However, if the new state is unsafe, then Pi must wait for *Request i* and the old resource-allocation state is restored.

An Illustrative Example

Consider a system with five processes Po through P4 and three resource types *A*, B, C. Resource type *A* has 10 instances, resource type *B* has 5 instances, and resource type C has 7 instances. Suppose that, at time *To*, the following snapshot of the system has been taken:

	Allocation	***Max***	***Available***
	A B C	*A B C*	*A B C*
Po	0 1 0	7 5 3	3 3 2
P1	2 0 0	3 2 2	
P2	3 0 2	9 0 2	
P3	2 1 1	2 2 2	
P4	0 0 2	4 3 3	

The content of the matrix *Need* is defined to **be *Max - Allocation*** and is

	Need		
	A	***B***	***C***
Po	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1

We claim that the system is currently in a safe state. Indeed, the sequence **<P1, *P3, P4, P2,* Po>** satisfies the safety criteria.

Suppose now that process P1 requests one additional instance of resource type *A* and two instances of resource type C, so *Request1* = (1,0,2). To decide whether this request can be immediately granted, we first check that *Request1*<= *Available (that* is, (1, 0, 2) <= (3, 3, 2)) which is true. We then pretend that this request has been fulfilled, and we arrive at the following new state:

	Allocation	*Need*	*Available*
	A B C	*A B C*	*A B C*
Po	0 1 0	7 4 3	2 3 0
P1	3 0 2	0 2 0	
P2	3 0 2	6 0 0	
P3	2 1 1	0 1 1	
P4	0 0 2	4 3 1	

We must determine whether this new system state is safe. To do so, we execute our safety algorithm and find that the sequence <P1, *P3, P4,* Po, *P2*> satisfies our safety requirement. Hence, we can immediately grant the request of process Pl.

You should be able to see, however, that when the system is in this state, a request for (3, 3, 0) by P

4 cannot be granted, since the resources are not available.

A request for (0, 2, 0) by Po cannot be granted, even though the resources are available, since the resulting state is unsafe.

2. Explain the necessary condition, which causes the deadlock situation to occur.

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

1. Mutual exclusion: At least one resource must be held in a non-sharable mode; that is, only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.

2. Hold and wait: A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.

3. No preemption: Resources cannot be preempted; that is, a resource can be released only voluntarily by the process holding it, after that process has completed its task.

4. Circular wait: A set (Po, P1, ..., *Pn)* of waiting processes must exist such that Po is waiting for a resource that is held by P1, Pl is waiting for a resource that is held by *P2,* ..., Pn-l is waiting for a resource that is held by *Pn,* and *Pn* is waiting for a resource that is held by Po.

3. Explain how you recover from the deadlock.

Recovery from Deadlock:

Process Termination

To eliminate deadlocks by aborting a process, we use one of two methods. In both methods, the system reclaims all resources allocated to the terminated processes.

- **Abort all deadlocked processes:** This method clearly will break the deadlock cycle, but at a great expense; these processes may have computed for a long time, and the results of these partial computations must be discarded and probably recomputed later.
- **Abort one process at a time until the deadlock cycle is eliminated**: This method incurs considerable overhead, since, after each process is aborted, a deadlock-detection algorithm must be invoked to determine whether any processes are still deadlocked.

Aborting a process may not be easy. If the process was in the midst of updating a file, terminating it will leave that file in an incorrect state. Similarly, if the process was in the midst of printing data on the printer, the system must reset the printer to a correct state before printing the next job.

Many factors may determine which process is chosen, including:

1. What the priority of the process is

2. How long the process has computed, and how much longer the process will compute before completing its designated task

3. How many and what type of resources the process has used (for example, whether the resources are simple to preempt)
4. How many more resources the process needs in order to complete
5. How many processes will need to be terminated?

2. Resource Preemption.

To eliminate deadlocks using resource preemption, we successively preempt some resources from processes and give these resources to other processes until the deadlock cycle is broken.

If preemption is required to deal with deadlocks, then three issues need to be addressed:

1. Selecting a victim: Which resources and which processes are to be preempted.

2. Rollback.: When a resource is preempted the process cannot continue its normal execution, hence rollback the process to some safe state and restart it from that state.

3. Starvation: In a system where victim selection is based primarily on cost factors, it may happen that the same process is always picked as a victim. As a result, this process never completes its designated task, a *starvation* situation that needs to be dealt with in any practical system

4. Write a Note on Resource allocation graph with an example.

Resource-Allocation Graph

Deadlocks can be described more precisely in terms of a directed graph called a system resource-allocation graph. **This graph consists of a set of vertices V and a set of edges E. The set of vertices V is partitioned into two different types of nodes P = {PI, P2, ..., Pn}, the set consisting of all the active processes in the system, and R = {Rl, R2, ..., Rm}, the set consisting of all resource types in the system.**

A directed edge from process Pi to resource type Rj is denoted by Pi⟶Rj; it signifies that process Pi requested an instance of resource type Rj and is currently waiting for that resource. A directed edge from resource type Rj to process Pi is denoted by Rj ⟶Pi; it signifies that an instance of resource type Rj has been allocated to process Pi. A directed edge Pi⟶Rj is called a **request edge;** a directed *edge Rj* ⟶*Pi* is called an **assignment edge**.

Pictorially, we represent each process Pi as a circle, and each resource type Rj as a square. Since resource type Rj may have more than one instance, we represent each such instance as a dot within the

square. Note that a *request* edge points to only the square Rj, whereas an assignment edge must also designate one of the dots in the square.

When process Pi *requests* an instance of resource type Rj, a *request* edge is inserted in the resource-allocation graph. When this *request* can be fulfilled, the *request* edge is *instantaneously* transformed to an assignment edge. .When the process no longer needs access to the resource it releases the resource, and as a result the assignment edge is deleted.

The resource-allocation graph shown in Figure depicts the following situation.

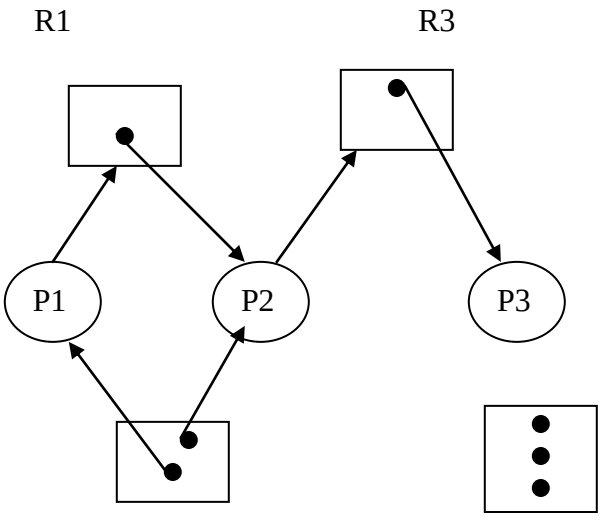
- **The sets P, R, and E:**
  P = {P1, P2, P3}
  R = {Rl, R2, R3, R4}
  E = {P1 Rl, P2 R3, Rl P2, R2 P2, R2 P1, R3 P3}

- **Resource instances:**
  One instance of resource type Rl
  Two instances of resource type R2
  One instance of resource type R3
  Three instances of resource type R4

- **Process states:**
  Processes P1 is holding an instance of resource type *R2,* and is waiting for an instance of resource type R1.

Process P2 is holding an instance of RI and *R2,* and is waiting for an instance of resource type *R3*

Process P3 is holding an instance of *R3.*



Resource-allocation graph

Given the definition of a resource-allocation graph, it can be shown that,

If the graph contains no cycles, then no process in the system is deadlocked. If the graph does contain a cycle, then a deadlock may exist.

If each resource type has exactly one instance, then a cycle implies that a deadlock has occurred. If the cycle involves only a set of resource types, each of which has only a single instance, then a deadlock has occurred. Each process involved in the cycle is deadlocked. In this case, a cycle in the graph is both a necessary and a sufficient condition for the existence of deadlock.

If each resource type has several instances, then a cycle does not necessarily imply that a deadlock has occurred. In this case, a cycle in the graph is a necessary but not a sufficient condition for the existence of deadlock.

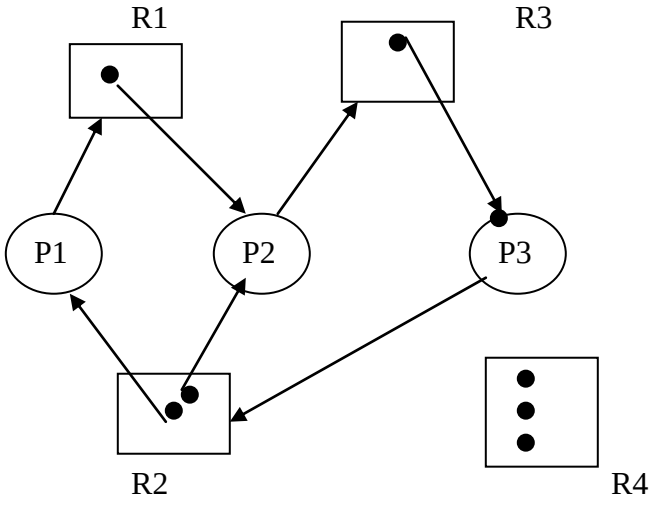
If one or more of the resource types involved in the cycle have more than one unit a knot is sufficient conditions for a deadlock occur.

Reachable Set: The reachable set of a node A is the set of all nodes B such that a path exists from A to B.

Knot: A knot is a nonempty set K of nodes such that the reachable set of each node in K is exactly the set K. A knot always contain one or more cycle.

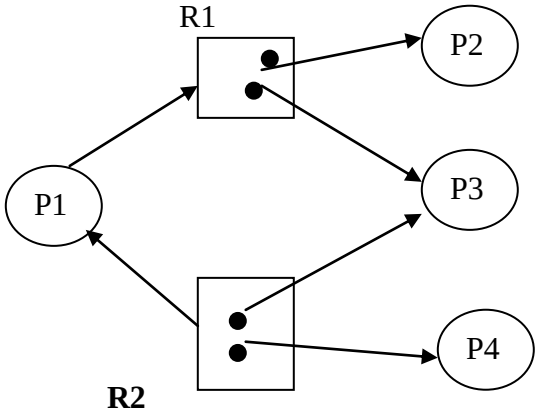
To illustrate this concept, let us return to the resource-allocation graph depicted in Figure. Suppose that process P3 requests an instance of resource type *R2*. Since no resource instance is currently available, a request edge P3 R2 is added to the graph (Figure). At this point, two minimal cycles exist in the system:

P1 - R1 - P2 - R3 - P3 - R2 - P1
P2 - R3 - P3 - R2 - P2



Resource-allocation graph with a deadlock.

Processes P1, *P2*, and P3 are deadlocked. Process P2 is waiting for the resource *R3*, which is held by process *P3*. Process *P3*, on the other hand, is waiting for either process P1 or process P2 to release resource *R2*. In addition, process P1 is waiting for process P2 to release resource R1.



Resource-allocation graph with a cycle but no deadlock.

Now consider the resource-allocation graph in Figure. In this example, we also have a cycle

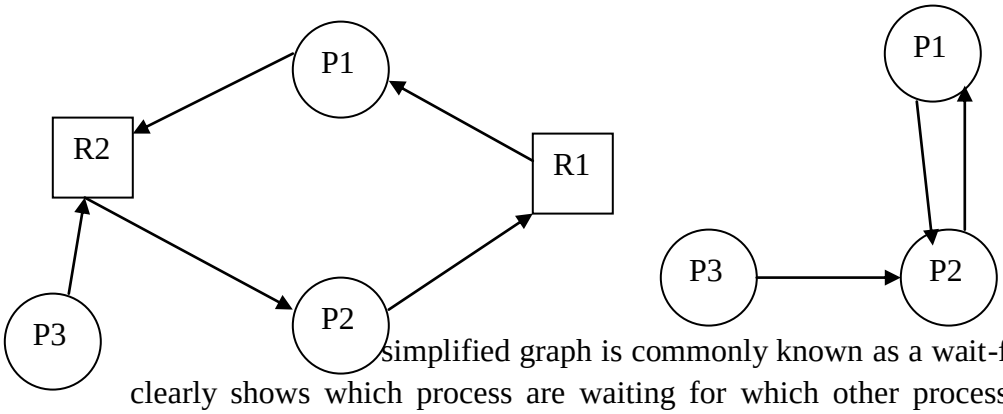
P1 R1 P3 R2 P1

However; there is no deadlock. Observe that process P4 may release its instance of resource type *R2*. That resource Can then be allocated to *P3*, breaking the cycle.

5. Write a Note on Wait for Graph with an example.

Wait-For Graph

When all the resource types have only a single unit each, a simplified from of resource allocation graph is normally used. The simplified graph is obtained from the original resource allocation graph by removing the resource nodes and collapsing the appropriate edges.



simplified graph is commonly known as a wait-for a graph (WFG) because it clearly shows which process are waiting for which other process. For instance in the above WFG, process P1 and P3 waiting for P2 and process P2 is waiting for P1. Since WFG is constructed only when each resource type has only a single unit, a cycle is both necessary and

sufficient condition for a deadlock in a WFG.

6. Explain Deadlock Prevention methods.

1. Mutual Exclusion

The mutual-exclusion conditions hold for non-sharable resources. For example, a printer cannot be simultaneously shared by several processes. **Sharable resources, on the other hand, do not require mutually exclusive access, and thus cannot be involved in a deadlock. Read only file are a good example of a sharable resource.** If several processes attempt to open a read-only file at the same time, they can be granted simultaneous access to the file. A process never needs to wait for a sharable resource. In general, however, we cannot prevent deadlocks by denying the mutual-exclusion condition: Some resources are intrinsically non-sharable.

2. Hold and Wait (All Request together or Collective requests)

To ensure that the hold-and-wait condition never occurs in the system, we must guarantee that, whenever a process requests a resource, it does not hold any other resources. **One protocol that can be used requires each process to request and be allocated all its resources before it begins execution. We can implement this provision by requiring that system calls requesting resources for a process precede all other system calls.**

The validity constraints on a resource requests is that a process must make its entire request together-typically at the starts of the execution.

These protocols have two main disadvantages. **First, resource utilization may be low**, since many of the resources may be allocated but unused for a long period. Consider two process P1 and P2 such that P1 needs a tape device and a printer, while P2 needs only printer. It is possible that P1 require a tape at the beginning of the execution and printer at the end of the execution. However it will forced to request both tape and printer at the beginning of the execution. This will simply idle the printer at the beginning of the execution.

Second, starvation is possible. A process that needs several popular resources may have to wait indefinitely, because at least one of the, resources that it needs is always allocated to some other process.

3. No Preemption

The third necessary condition is that there be no preemption of resources that have already been allocated. To ensure that this condition does not hold, we can use the following protocol. **If a process is holding some resources and requests another resource that cannot be immediately allocated to it (that is, the process must wait), then all resources currently being held are preempted.** In other words, these resources are implicitly released. The preempted resources are added to the list of resources for which the process is waiting. The process will be restarted only when it-can regain its

4 Circular Wait (Resource Ranking)

The fourth and final condition for deadlocks is the circular-wait condition. **Wait to ensure that this condition never holds is to impose a total ordering of all resource types, and to require that each process requests resources in an increasing order of enumeration.**
Let R = {R1, R2, ..., *Rm}* be the set of resource types. We assign to each resource type a unique integer number, which allows us to compare two resources and to determine whether one precedes another in our ordering.
F (tape drive) = 1,
F (disk drive) = 5,
F (printer) = 12.

We can now consider the following protocol to prevent deadlocks: Each process can request resources only in an increasing order of enumeration that is, a process can initially request any number of instances of a resource type, say *Ri.* After that, the process can request instances or resource type Rj if and only if *F (Rj) > F (Ri).* If several instances of the same recourse type are needed, a *single* request for all of them must be issued. For example, using the function defined previously, a process that wants to use the tape drive and printer at the same time must first request the tape drive and then request the printer.

Alternatively, we can require that, whenever a process requests an instance of resource type *Rj*, it has released any resources *Ri* such that *F (Ri) >= F (Rj).*

We can demonstrate this fact by assuming that a circular wait exists (proof by contradiction). Let the set of processes involved in the circular wait be {Po, P1, ..., *Pn}*, where *Pi* is waiting for a resource *Ri,* which is held by process Pi+l. Then, since process Pi+l is holding resource *Ri* while requesting resource Ri+l, we must have *F (Ri)* < F (Ri+l), for all *i.* But this condition means that F (Ro) < F (Rl) < ... < *F (Rn)* < F (Ro). By transitivity, *F (Ro)* < F (Ro), which is impossible. Therefore, there can be no circular wait.

7. Explain Deadlock Detection using Single Instance of Each Resource Type.

Single Instance of Each Resource Type

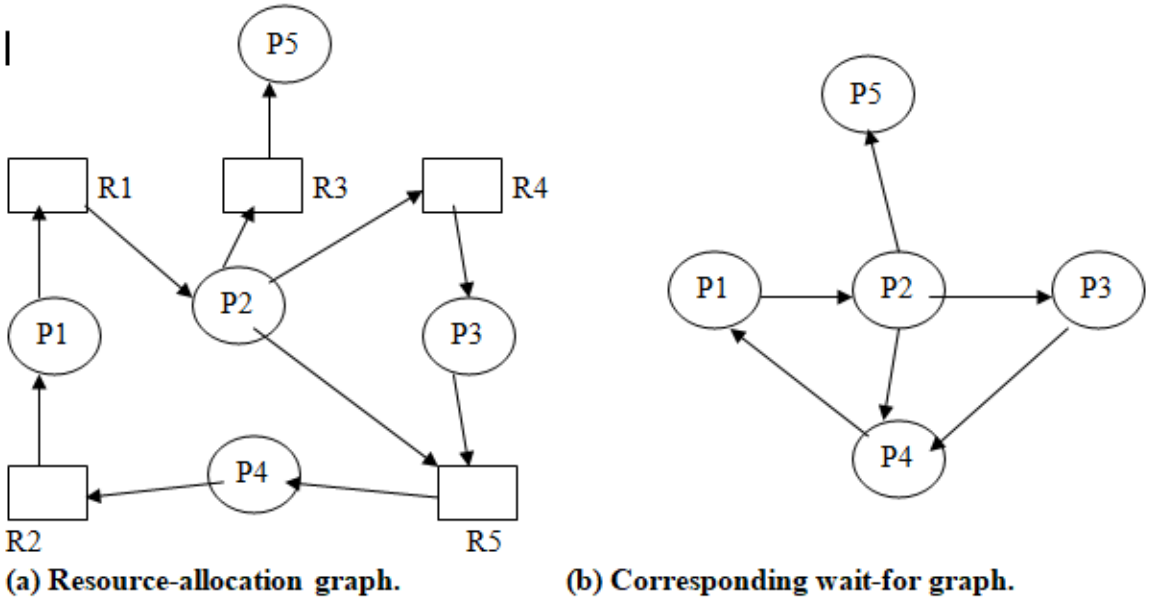
If all resources have only a single instance, then we can define a deadlock detection algorithm that uses a variant of the resource-allocation graph, called a ***wait-for* graph**. We obtain this graph from the resource allocation graph by removing the nodes of type resource and collapsing the appropriate edges.

More precisely, an edge from *Pi* to Pj in a wait-for graph implies that process *Pi* is waiting for process Pj to release a resource that *Pi* needs. An edge *Pi* ---> Pj exists in await-for graph if and only if the corresponding resource allocation graph contains two edges *Pi* ---> *Rq* and *Rq* ---> Pj for some

resource *Rq*. For example, in Figure we present a resource-allocation graph and the corresponding wait-for graph.

As before, a deadlock exists in the system if and only if the wait-for graph contains a cycle. To detect deadlocks the system needs to *maintain* the wait-for graph and periodically to *invoke algorithm* that searches for a cycle in the graph.

An algorithm to detect a cycle in a graph requires an order of n^2 operations, where *n* is the number of vertices in the graph.



(a) Resource-allocation graph. **(b) Corresponding wait-for graph.**

8. Explain Deadlock Detection using Several Instances of a Resource Type.

Several Instances of a Resource Type

The wait-for graph scheme is not applicable to a resource-allocation system with multiple instances of each resource type. The deadlock-detection algorithm that we describe next is applicable to such a system. The algorithm employs several time-varying data structures that are similar to those used in the banker's algorithm:

. Available: A vector of length, *m* indicates the number of available resources of each type.

. Allocation: An *n* x *m* matrix defines the number of resources of each type currently allocated to each process.

. Request: An *n* x *m* matrix indicates the current request of each process. If *Request [i,j] = k*, then process Pj is requesting *k* more instances of resource type *Rj*.

1. Let *Work* and *Finish* be vectors of length *m* and *n*, respectively.

Initialize *Work: = Available.* For *i* = 1, 2, ..., *n,*
If *Allocation i <> 0* then *Finish[i]* := *false;*
Otherwise, *Finish[i]:= true.*
2. Find an index *i* such that both
a. *Finish[i] = false.*
b. *Request i<= Work.*
If no such *i* exists, go to step 4.

3. *Work: = Work + Allocation;*
Finish[i]:= true
Go to step 2.

4. If *Finish[i]* = false, for some *i,* 1<= *i*<= *n,* then the system is in a deadlock state. Moreover, if *Finish[i] = false,* then process Pi is deadlocked.

This algorithm requires an order of *m* x n^2 operations to detect whether the system is in a deadlocked state.

To illustrate this algorithm, we consider a system with five processes Po through P4 and three resource types *A,* B, C. Resource type *A* has 7 instances, resource type B has 2 instances, and resource type C has 6 instances. Suppose that, at time *To,* we have the following resource-allocation state:

	Allocation	*Request*	*Available*
	A B C	*A B C*	*A B C*
Po	*0 1 0*	*0 0 0*	0 0 0
P1	*2 0 0*	2 0 2	
P2	3 0 3	*0 0 0*	
P3	2 1 1	*1 0 0*	
P4	0 0 2	0 0 2	

We claim that the system is not in a deadlocked state. Indeed, if we execute our algorithm, we will find that the sequence **<Po, *P2, P3,* P1, *P4*>** will result in *Finish[i]* = true for all i.

Suppose now that process P2 makes one additional request for an instance of type C. The *Request* matrix is modified as follows:

	Request		
	A	B	C
P0	0	0	0
P1	2	0	2
P2	0	0	1
P3	1	0	0
P4	0	0	2

We claim that the system is now deadlocked. Although we can reclaim the resources held by process Po, the number of available resources is not sufficient to fulfill the requests of the other processes. Thus, a deadlock exists, consisting of processes P1, P2, P3 and P 4.

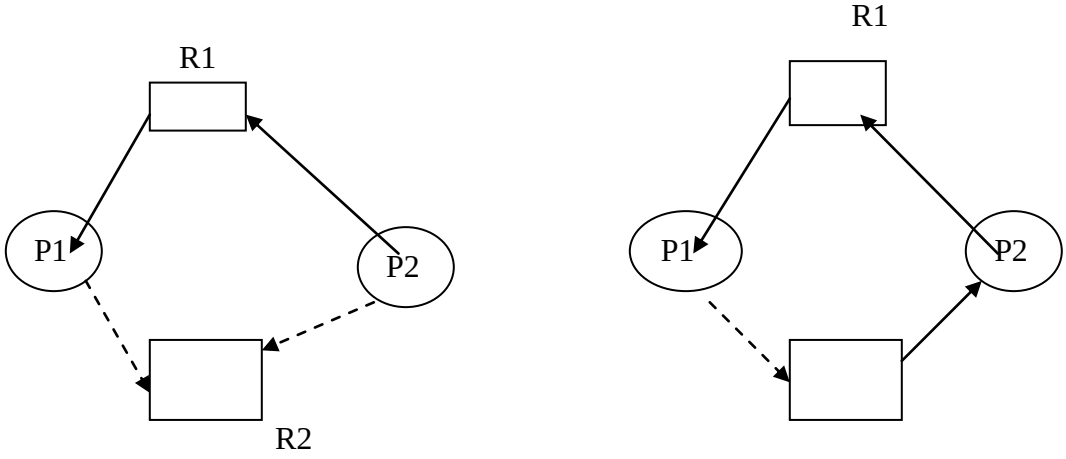
9. Explain Deadlock Avoidance using Resource-Allocation Graph Algorithm.

Resource-Allocation Graph Algorithms

If we have a resource-allocation system with only one instance of each resource type, a variant of the resource-allocation graph can be used for deadlock avoidance.

In addition to the request and assignment edges, we introduce a new type of edge, **called a claim edge.** A claim edge Pi *Ri* indicates that process Pi may request resource Rj at some time in the future. This edge resembles request edge in direction, but is represented by a dashed line. When process *Pi* requests resource *Rj*, the claim edge Pi *Rj* is concerted to a request edge. Similarly, when a resource *Rj* is released by *Pi*, *a*ssignment edge *Rj* *Pi* *i*s reconverted to a claim edge *Pi* *Rj*. We note that the resources must be claimed a priori in the system. That is, before process *Pi* starts executing, all its claim edges must already appear in the resource-allocation graph. We can relax this condition by allowing a claim edge. *Pi* *Rj* to be added to the graph only if all the edges associated with process *Pi* is claim edges.

Suppose that process *Pi* requests resource *Rj*. The request can be granted only if converting the request edge *Pi* *Rj* to an assignment edge *Rj* *Pi* does not result in the formation of a cycle in the resource-allocation graph. Note that we check for safety by using a cycle-detection algorithm. An algorithm for detecting a cycle in this graph requires an order of n^2 operations, where *n* is the number of processes in the system.



Resource-allocation graph for deadlock avoidance. R2

If no cycle exists, then the allocation of the resource will leave the system in a safe state. If a cycle is found, then the allocation will put the system in an unsafe state. Therefore, process *Pi* will have to wait for its requests to be satisfied.

To illustrate this algorithm, we consider the resource-allocation graph of Figure 1. Suppose that P2 requests *R2*. Although R2 is currently free, we cannot allocate it to *P2*, since this action will create a

cycle in the graph (Figure 2). A cycle indicates that the system is in an unsafe state. If P1 requests *R2,* and P2 requests R1, then a deadlock will occur.

One Mark Questions:

1. What is a meant by a Logical Address?

An address generated by the CPU is commonly referred to as a logical address,

2. What is a meant by a Physical Address?

An address seen by the memory unit-that is, the one loaded into the memory-address register of the **memory-is commonly referred to as a physical address**.

3. What is a meant by a Logical Address Space?

The set of all addresses generated by a program is a **logical-address space.**

4. What is a meant by a Physical Address Space?

The **set of all physical addresses corresponding to these logical addresses is a physical-address space**.

5. What is Swapping?

Swapping is the technique of temporarily removing inactive programs from the memory of a computer system.

6. What is a paging?

Paging is a memory-management scheme that permits the physical-address space of a process to be noncontiguous.

7. What is Segmentation?

An important aspect of memory management that became unavoidable with paging is the separation of the user's view of memory and the actual physical memory. The user's view of memory is not the same as the actual physical memory. The user's view is mapped onto physical memory. **The mapping allows differentiation between logical memory and physical memory.**

8. The set of all logical address generated by a program is referred to as…… **logical-address space.**

9. State True or False

Worst fit allocates the largest hole - True

10. What is a Compaction?

Compaction is the process of physical shifting of the process from one part of the memory to another part of the memory.

11. State True or False

Compaction is the process is used to overcome the problem of external fragmentation – True

12. Define the term fragmentation.

Memory fragmentation implies the existence of unusable memory areas in a computer system. Early Multiprogramming systems used a partitioned approach to memory management. Each memory partitioned was a single contiguous area in the memory. Memory allocation is at the beginning of the execution.

13. What is internal fragmentation?

Internal fragmentation arises in the system when memory area allocated to a program is not fully utilized by it.

14. What is External fragmentation?

External fragmentation arises in the system when free memory area existing in a system is too small to be allocating to a job.

15. The run time mapping from virtual to physical address is done by hardware device called…… Memory Management Unit (MMU)

16. What is a Page Table?

The page number is used as an index into a page table. The page table contains the base address of each page in physical memory.

17. What is a Segment Table?

Segment Table is a table that stores the information about each segment of the process. First column stores the size or length of the segment. Second column stores the base address or starting address of the segment in the main memory.

Five Marks Questions

1. What do you mean by internal and external fragmentation? How can we solve the problem of fragmentation? Explain

Internal fragmentation arises in the system when memory area allocated to a program is not fully utilized by it.

There are three jobs A, B and C exits in a system's allocate 100KB of memory to Job But Job A utilized only 80KB of memory through out its execution.20KB of memory allocated to Job A becomes unused . It cannot be allocate to any other jobs. This is internal fragmentation**.**

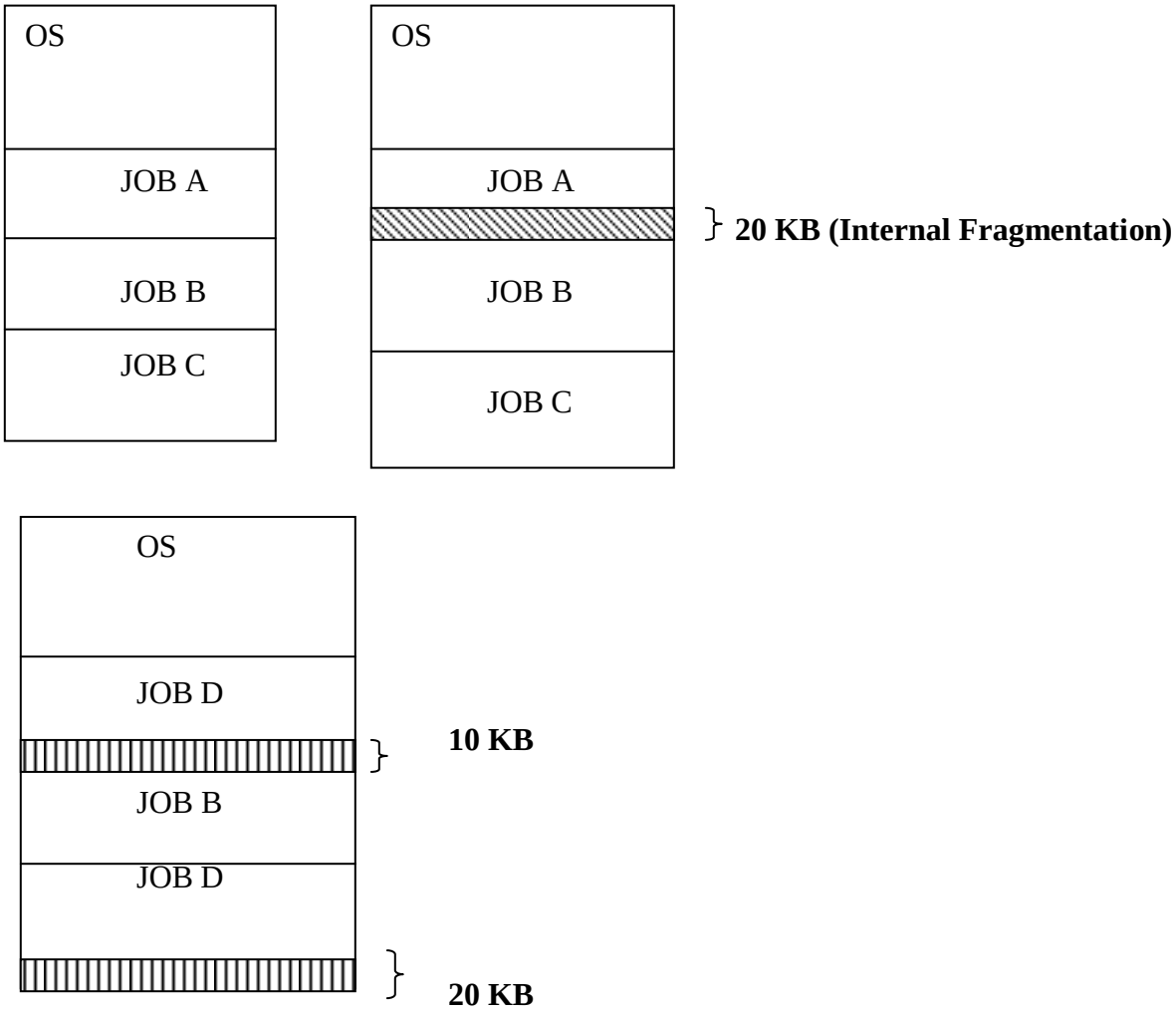
External fragmentation arises in the system when free memory area existing in a system is too small to be allocating to a job.

Let Job A complete and Job D whose memory requirement is 90KB is placed in its place. An area of 10Kb of free memory area exists between Job D and B. However this cannot be used because it is too small to accommodate a job. Now let Job C complete and let Job E be placed in its location. This leads another free area.(20KB). A total of 30KB of unallocated memory now exists in the system but a job, say F requiring 25KB cannot be initiated because a single contiguous free area 25KB is not available. Hence 10Kb and 20KB are unusable. This is called external fragmentation.

One solution to the problem of external fragmentation is compaction. The goal is to shuffle the memory contents to place all free memory together in one large block.

Compaction is the process of physical shifting of the process from one part of the memory to another part of the memory.

Another possible solution to the external-fragmentation problem is to permit the logical-address space of a process to be noncontiguous, thus allowing a process to be allocated physical memory wherever the latter is available. **Two complementary techniques achieve this solution: paging and segmentation.**



(10 +20)KB External Fragmentation

2. Explain Segmentation with an example.

Segmentation

An important aspect of memory management that became unavoidable with paging is the separation of the user's view of memory and the actual physical memory. The user's view of memory is not the same as the actual physical memory. The user's view is mapped onto physical memory. **The mapping allows differentiation between logical memory and physical memory.**

Basic Method

We think of it as a main program with a set of subroutines, procedures, functions, or modules. There may also be various data structures: tables, arrays, stacks, variables, and so on. Each of these modules or data elements is referred to by name. We talk about "the symbol table," "function *Sqrt,"* "the main program," without caring what addresses in memory these elements occupy. You are not concerned with whether the symbol table is stored before or after the *Sqrt* function. Each of these segments is of variable length; the length is intrinsically defined by the purpose of the segment in the program. Elements within a segment are identified by their offset from the beginning of the segment:

Segmentation is a memory-management scheme that supports this user view of memory. A logical-address space is a collection of segments. Each segment has a name and a length. The addresses specify both the segment name and the offset within the segment. The user therefore specifies each address by two quantities: a segment name and an offset.

For simplicity of implementation, segments are numbered and are referred to by a segment number, rather than by a segment name. Thus, a logical address consists of a *two tuple:*

<Segment-number, offset>.

Normally, the user program is compiled, and the compiler automatically constructs segments reflecting the input program. A Pascal compiler might create separate segments for the following:

1. The global variables;
2. The procedure call stack, to store parameters and return addresses;
3. The code portion of each procedure or function;
4. The local variables of each procedure and function.

3. Explain Paging with an example.

Paging

Paging is a memory-management scheme that permits the physical-address space of a process to be noncontiguous. Paging avoids the considerable problem of fitting the varying-sized memory chunks onto the backing store, from which most of the previous memory-management schemes suffered. When some code fragments or data residing in main memory need to be swapped out, space must be found on the backing store. The fragmentation problems discussed in connection with main memory are also prevalent with backing store, except that access is much slower, so compaction is impossible.

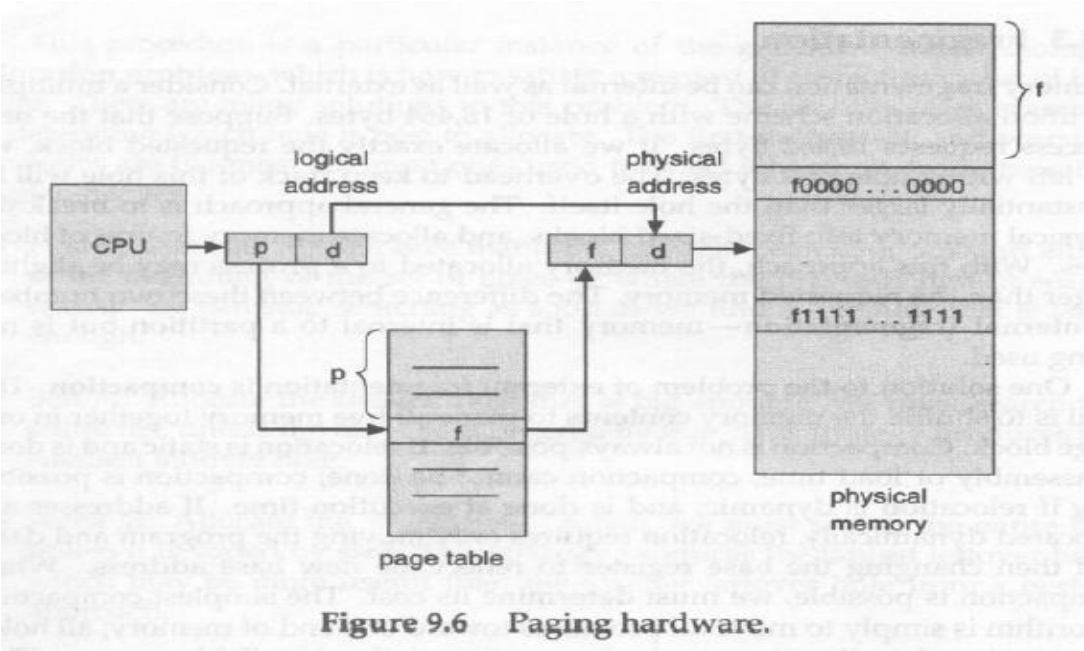


Figure 9.6 Paging hardware.

Physical memory is broken into fixed-sized blocks called frames. **Logical memory is also broken into blocks of the same size called pages**. When a process is to be executed, its pages are loaded into any available memory frames from the backing store. The backing store is divided into fixed-sized blocks that are of the same size as the memory frames.

The hardware support for paging is illustrated in Figure. Every address generated by the CPU is divided into two parts: a **page number (p) and a page offset (d).** The page number is used as an index into a page table. The page table contains the base address of each page in physical memory. This base address is combined with the page offset to define the physical memory address that is sent to the memory unit. The paging model of memory is shown in Figure.

The page size (like the frame size) is defined by the hardware. The size of a page is typically a power of 2, varying between 512 bytes and 16 MB per page, depending on the computer architecture. The selection of a power of 2 as a page size makes the translation of a logical address into- a page number and page offset particularly easy. If the size of logical-address space is 2^m, and a page size is 2^n addressing units (bytes or words), then the high-order ***m - n* bits of a logical address** designate the page number, and the ***n* low-order bits designate the page offset**. Thus, the logical address is as follows:

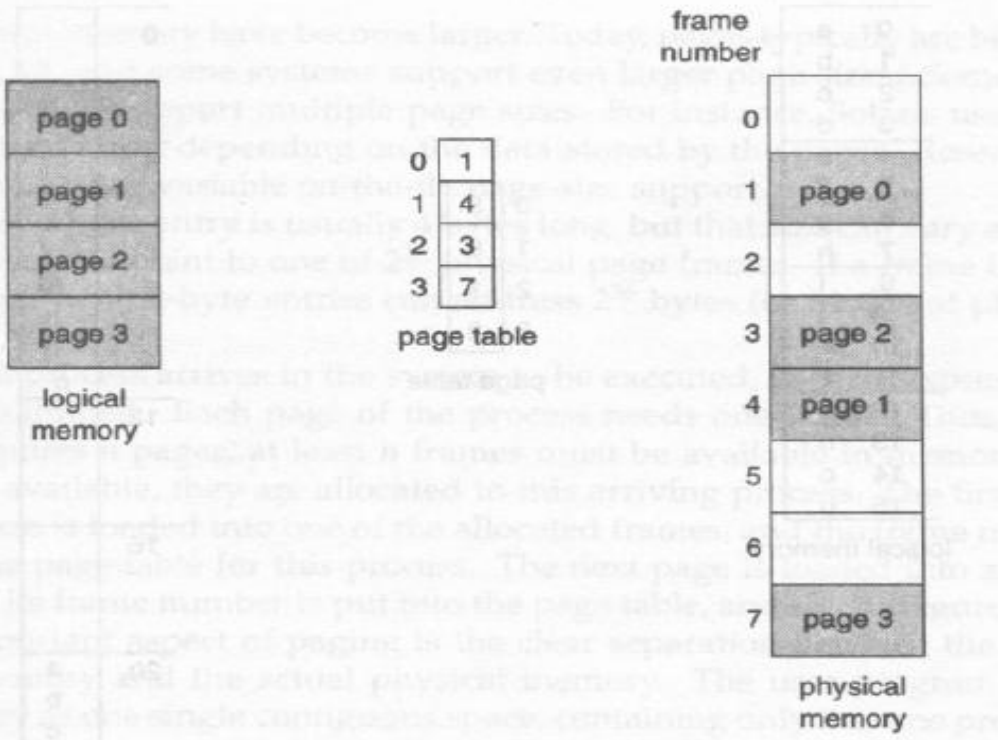
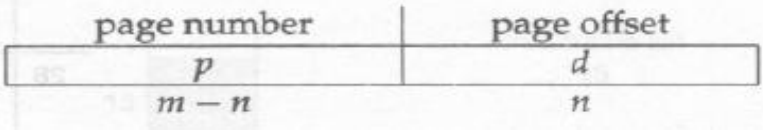


Figure 9.7 Paging model of logical and physical memory.



Where ***p* is an index into the page table and *d* is the displacement within the page.**

As a concrete (although minuscule) example, consider the memory in Figure. Using a page size of 4 bytes and a physical memory of 32 bytes (8 pages), we show how the user view of memory can be mapped into physical memory.

Logical address 0 is page 0, offset O. indexing into the page table; we find that page 0 is in frame 5. Thus, logical address 0 maps to physical address 20 (= (5 x 4) + 0).

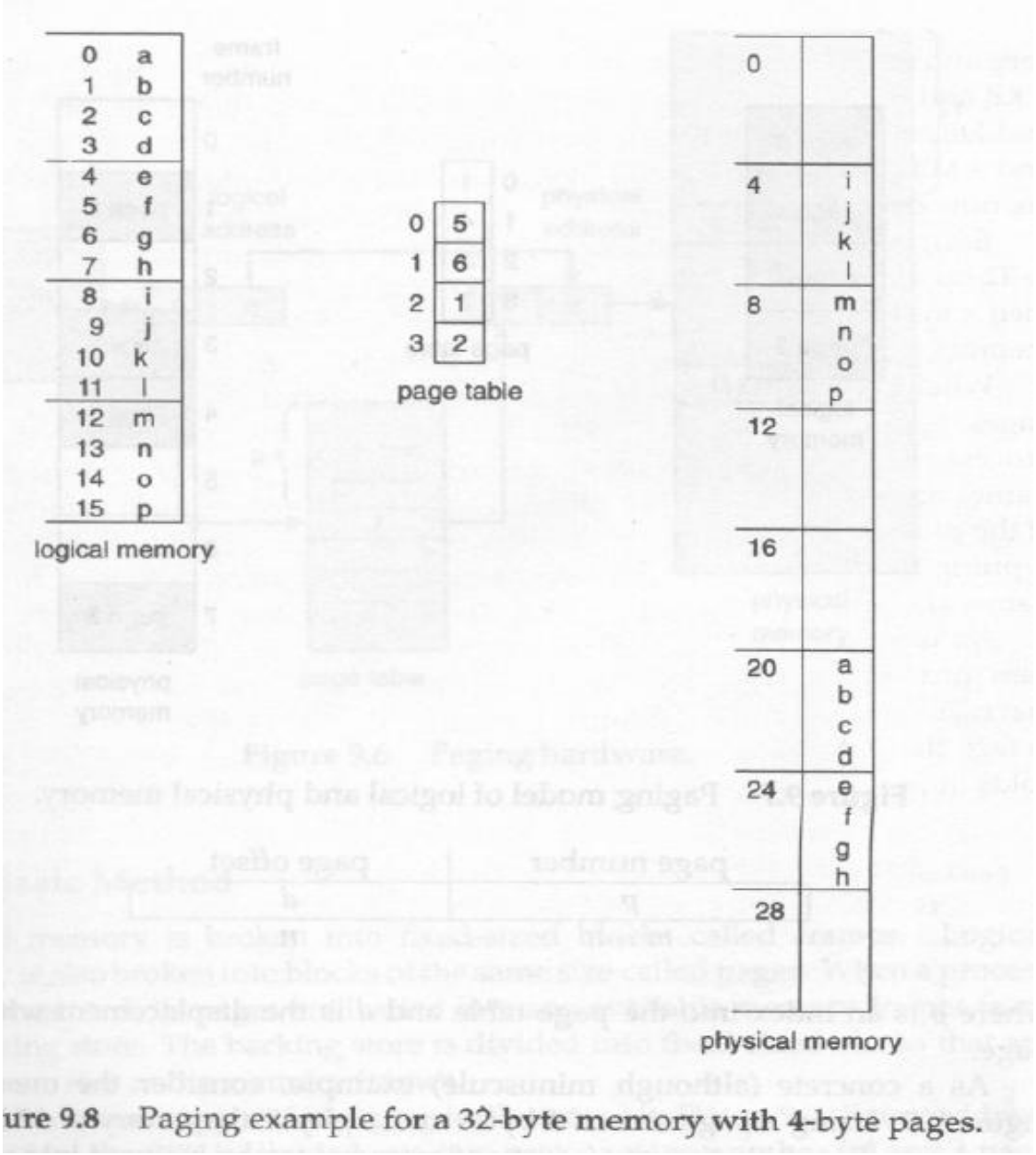
Logical address 3 (page 0, offset 3) maps to physical address 23 (= (5 x 4) + 3). Logical address 4 is page 1, offset 0; according to the page table, page 1 is mapped to frame 6.

Thus, logical address 4 maps to physical address 24 (= (6 x 4) + 0). Logical address 13 maps to physical address 9.

You may have noticed that paging itself is a form of dynamic relocation. Every logical address is bound by the paging hardware to some physical address. Using paging is similar to using a table of base (or relocation) registers, one for each frame of memory.

When we use a paging scheme, we have no external fragmentation: *Any* free frame can be allocated to a process that needs it. However, we may have some internal fragmentation. Notice that frames are allocated as units. If the memory requirements of a process do not happen to fall on page boundaries, the *last* frame allocated may not be completely full. For example, if pages are 2,048 bytes, a process of 72,766 bytes would need 35 pages plus 1,086 bytes. It would be allocated 36 frames, resulting in an internal fragmentation of 2048 -1086 = 962 bytes. In the worst case, a process would need *n* pages plus

one byte. It would be allocated $n + 1$ frames, resulting in an internal fragmentation of almost an entire frame.



ure 9.8 Paging example for a 32-byte memory with 4-byte pages.

4. What do you mean by Swapping? Explain.

Swapping

A process needs to be in memory to be executed. **A process, however, can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution**.

Swapping is the technique of temporarily removing inactive programs from the memory of a computer system.

For example, assume a Time sharing environment with a round-robin CPU-scheduling algorithm. When a quantum expires, the memory manager will start to swap out the process that just finished, and to swap in another process to the memory space that has been freed (Figure). In the meantime, the CPU scheduler will allocate a time slice to some other process in memory. When each process finishes its quantum, it will be swapped with another process. Ideally, the memory manager can swap processes fast enough that some processes will be in memory, ready to execute, when the CPU scheduler wants to reschedule the CPU. The quantum must also be sufficiently large that reasonable amounts of computing are done between swaps.

5. Explain Contiguous Memory allocation.

Contiguous Memory Allocation

The main memory must accommodate both the operating system and the various user processes. We therefore need to allocate different parts of the main memory in the most efficient way possible. This section will explain one method, contiguous memory allocation.

We usually want several user processes to reside in memory at the same time. We therefore need to consider how to allocate available memory to the processes that are in the input queue waiting to be brought into memory. **In this contiguous memory allocation, each process is contained in a single contiguous section of memory. In contiguous memory allocation all the set of code and data about a process is stored in a single contiguous area.**

6. Explain Logical and Physical Address space.

An address generated by the CPU is commonly referred to as a logical address, whereas an address seen by the memory unit-that is, the one loaded into the memory-address register of the **memory-is commonly referred to as a physical address**.

The compile-time and load-time address-binding methods generate identical logical and physical addresses. However, the execution-time addresses binding scheme results in differing logical and physical addresses. In this case, we usually refer to the logical address as a virtual address. The set of all logical addresses generated by a program is a **logical-address space**; the **set of all physical addresses corresponding to these logical addresses is a physical-address space**. Thus, in the execution-time address-binding scheme, the logical- and physical-address spaces differ.

One Marks Questions

1. What is hit ratio?

The performance of memory is frequently measured in terms of quantity is called hit ratio. When the CPU needs to find the word in the cache, if the word is found in the cache then it's a hit. If the word is not found in the cache, it is in main memory as counted miss. The ratio of number of hits is divided by the total CPU reference of memory is called hit ratio.

2. What is Demand Paging?

Demand Paging

A demand-paging system is similar to a paging system with swapping. Processes reside on secondary memory (which is usually a disk). When we want to execute a process, we swap it into memory.

3. What is a Virtual Memory?

Virtual memory is a technique that allows the execution of processes that may not be completely in memory.

4. What is meant by Local Allocation of frames?

Local replacement requires that each process select from only its own set of allocated frames.

5. What is meant by Global Allocation of frames?

global replacement and local replacement. **Global replacement allows a process to select a replacement frame from the set of all frames, even if that frame is currently allocated to some other process; one process can take a frame from another.**

6. LFU stands for… **Least Frequently Used** ……..
7. MFU stands for…… **Most Frequently Used** …….

Five Marks Questions

1. Explain any TWO page replacement algorithm.

1. FIFO Page Replacement

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

For our example reference string, our three frames are initially empty. The first three references (7, 0,1) cause page faults, and are brought into these empty frames. The next reference (2) replaces page 7,

because page 7 was brought in first. Since 0 is the next reference and 0 is already in memory, we have no fault for this reference. The first reference to 3 results in page 0 being replaced, since it was the first of the three pages in memory (0, 1, and 2) to be brought in. Because of this replacement, the next reference, to 0, will fault. Page 1 is then replaced by page O. This process continues as shown in Figure. Every time a fault occurs, we show which pages are in our three frames. There are 15 faults altogether.

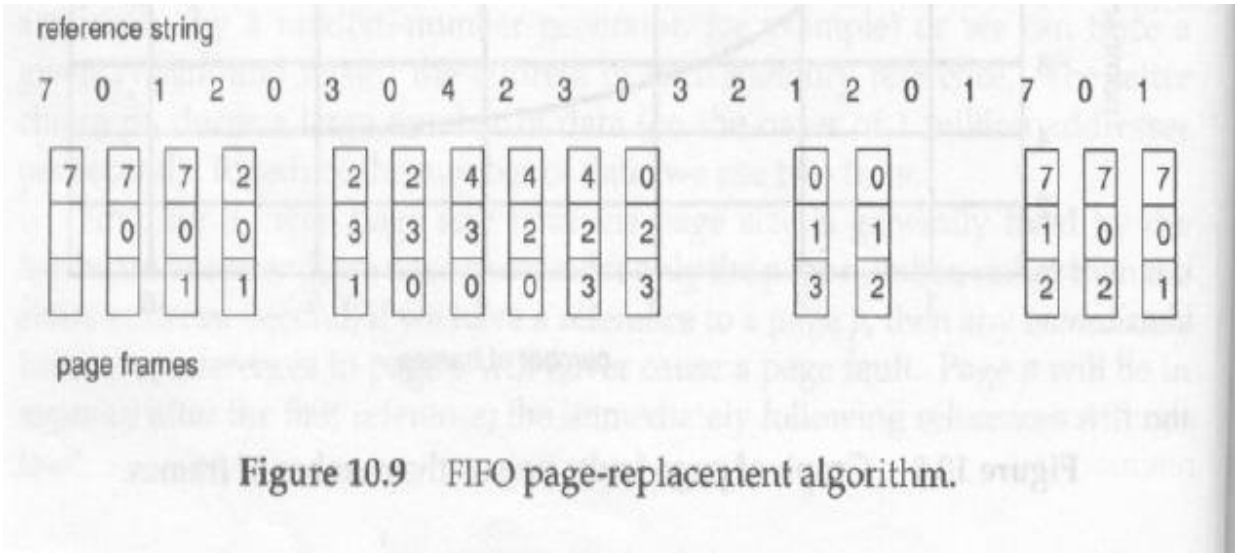


Figure 10.9 FIFO page-replacement algorithm.

To illustrate the problems that are possible with a FIFO page-replacement algorithm, we consider the reference string

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

Figure shows the curve of page faults versus the number of available frames. We notice that the number of faults for four frames (10) is *greater* than the number of faults for three frames (nine)! This most unexpected result is known as **Belady's anomaly**: For some page-replacement algorithms, the page fault rate may *increase* as the number of allocated frames increases.

2. Optimal Page Replacement

One result of the discovery of Belady's anomaly was the search for an optimal page-replacement algorithm. An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms, and will never suffer from Belady's anomaly. It is simply**Replace the page that will not be used for the longest period of time.**

Use of this page-replacement algorithm guarantees the lowest possible page fault rate for a fixed number of frames.

For example, on our sample reference string, the optimal page-replacement algorithm would yield nine page faults, as shown in Figure .The first three references cause faults that fill the three empty frames. The reference to page 2 replaces page 7, because 7 will not be used until reference 18, whereas page 0 will be used at 5, and page 1 at 14. The reference to page 3 replaces page 1, as page 1 will be the last of the three pages in memory to be referenced again. With only nine page faults, optimal replacement is much better than a FIFO algorithm, which had 15 faults. In fact, no replacement algorithm can process this reference string in three frames with less than nine faults.

Unfortunately, the optimal page-replacement algorithm is difficult to implement, because it

requires future knowledge of the reference string.

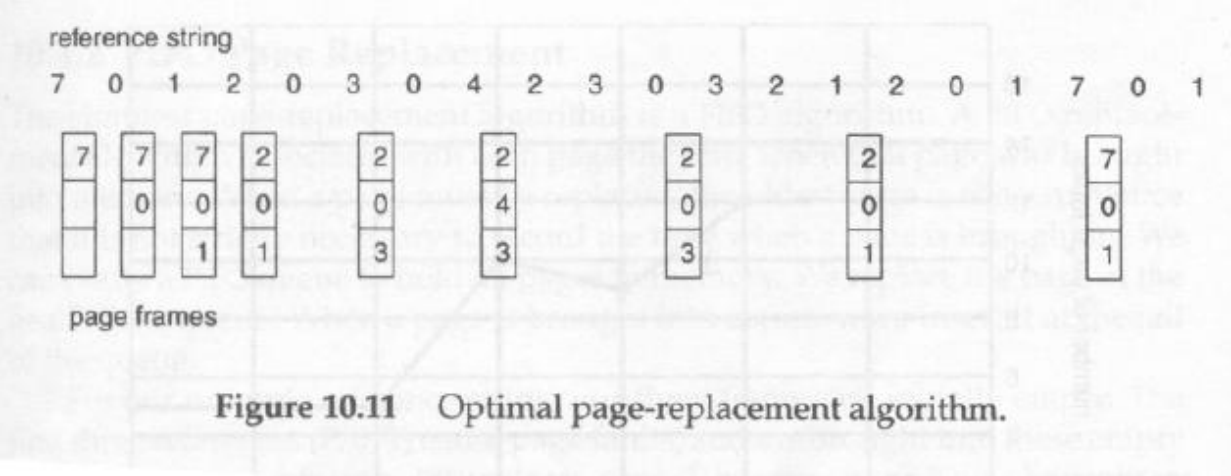


Figure 10.11 Optimal page-replacement algorithm.

2. Explain Demand Paging with a diagram.

Demand Paging

A demand-paging system is similar to a paging system with swapping. Processes reside on secondary memory (which is usually a disk). When we want to execute a process, we swap it into memory. Rather than swapping the entire process into memory, however, we use a lazy swapper. A lazy swapper never swaps a page into memory unless that page will be needed.

With this scheme, we need some form of hardware support to distinguish between those pages that are in memory and those pages that are on the disk. **The valid-invalid bit scheme described in this Section can be used for this purpose. This time, h**

wever, when this bit is set to "valid," this value indicates that the associated page is both legal and in memory. If the bit is set to "invalid," this value indicates that the page either is not valid is currently on the disk. The page-table entry for a page that is brought into memory is set as usual, but the page-table entry for a page that is not currently in memory is simply marked invalid, or contains the address of the page on disk.

3. Explain FIFO page replacement algorithm with an example.

1. FIFO Page Replacement

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

For our example reference string, our three frames are initially empty. The first three references (7, 0,1) cause page faults, and are brought into these empty frames. The next reference (2) replaces page 7, because page 7 was brought in first. Since 0 is the next reference and 0 is already in memory, we have no fault for this reference. The first reference to 3 results in page 0 being replaced, since it was the first of the three pages in memory (0, 1, and 2) to be brought in. Because of this replacement, the next reference, to 0, will fault. Page 1 is then replaced by page O. This process continues as shown in Figure. Every time a fault occurs, we show which pages are in our three frames. There are 15 faults altogether.

- The FIFO page-replacement algorithm is easy to understand and program.
- However, its performance is not always good. The page replaced may be an initialization module that was used a long time ago and is no longer needed. **On the other hand, it could contain a heavily used variable that was initialized early and is in constant use.**

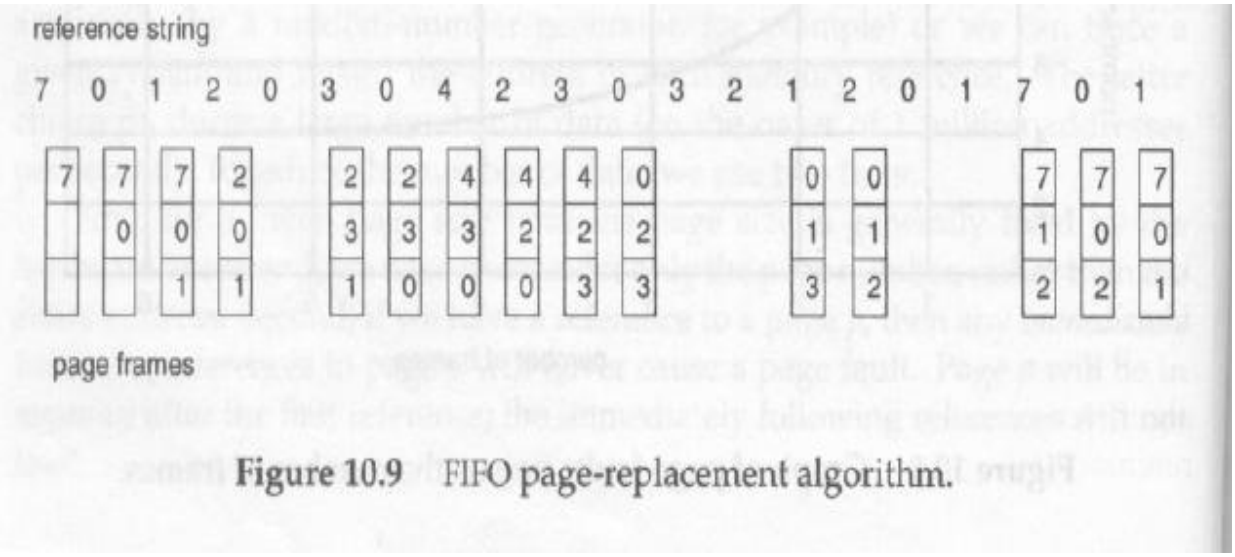


Figure 10.9 FIFO page-replacement algorithm.

To illustrate the problems that are possible with a FIFO page-replacement algorithm, we consider the reference string

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

Figure shows the curve of page faults versus the number of available frames. We notice that the number of faults for four frames (10) is *greater* than the number of faults for three frames (nine)! This most unexpected result is known as **Belady's anomaly**: For some page-replacement algorithms, the page fault rate may *increase* as the number of allocated frames increases.

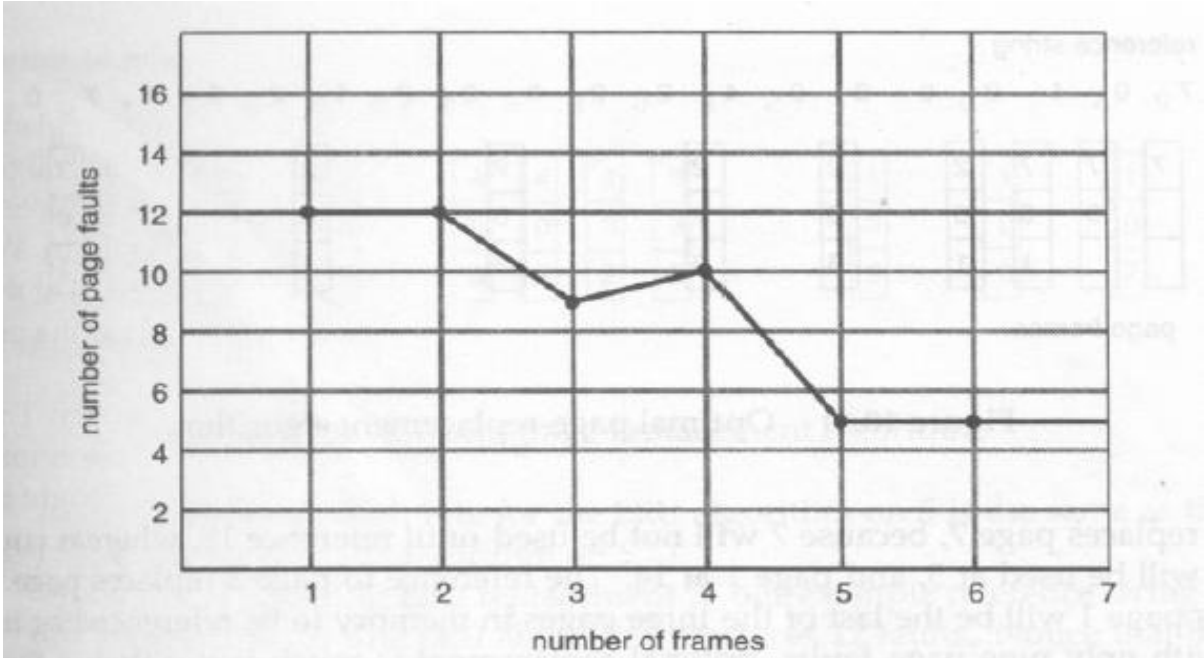


Figure 10.10 Page-fault curve for FIFO replacement on a reference string.

4. Explain optimal page replacement algorithm with an example.

One result of the discovery of Belady's anomaly was the search for an optimal page-replacement algorithm. An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms, and will never suffer from Belady's anomaly. It is simply**Replace the page that will not be used for the longest period of time.**

Use of this page-replacement algorithm guarantees the lowest possible page fault rate for a fixed number of frames.

For example, on our sample reference string, the optimal page-replacement algorithm would yield nine page faults, as shown in Figure .The first three references cause faults that fill the three empty frames. The reference to page 2 replaces page 7, because 7 will not be used until reference 18, whereas page 0 will be used at 5, and page 1 at 14. The reference to page 3 replaces page 1, as page 1 will be the last of the three pages in memory to be referenced again. With only nine page faults, optimal replacement is much better than a FIFO algorithm, which had 15 faults. In fact, no replacement algorithm can process this reference string in three frames with less than nine faults.

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

5. Explain LRU page replacement algorithm with an example.

. LRU Page Replacement

If we use the recent past as an approximation of the near future, then we will replace the page that *has not been used* for the longest period of time (Figure). This approach is the least-recently-used (LRU) algorithm.

LRU replacement associates with each page the time of that pages last use. **When a page must be replaced, LRU chooses that page that has not been used for the longest period of time**. This strategy is the optimal page-replacement algorithm looking backward in time, rather than forward.

The result of applying LRU replacement to our example reference string is shown in Figure. The LRU algorithm produces 12 faults. Notice that the first five faults are the same as the optimal replacement. When the reference to page 4 occurs, however, LRU replacement sees that, of the three frames in memory, page 2 was used least recently. The most recently used page is page 0 and just before that page 3 was used. Thus the LRU algorithm replaces page 2 not knowing that page 2 is about to be used. When it then faults for page 2 the LRU algorithm replaces page 3 since, of the three pages in memory {0, 3, 4}, page 3 is the least recently used. Despite these problems, LRU replacement with 12 faults is still much better than FIFO replacement with 15.

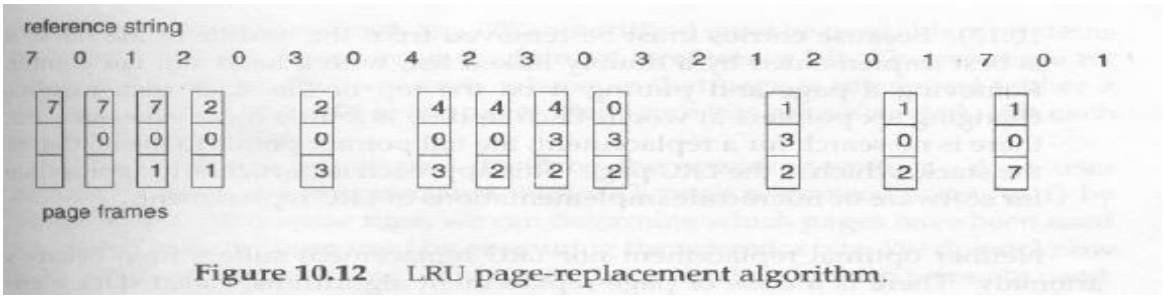


Figure 10.12 LRU page-replacement algorithm.

6. Explain LRU Approximation Page Replacements algorithm.

LRU Approximation Page Replacements

Few computer systems provide sufficient hardware support for true LRU page replacement. Some systems provide no hardware support, and other page replacement algorithms (such as a FIFO algorithm) must be used. Many systems provide some help, however, in the form of **a reference bit**.

The reference bit for a page is set, by the hardware, whenever that page is referenced (either a read or a write to any byte in the page). Reference bits are associated with each entry in the page table.

Initially, all bits are cleared (to 0) by the operating system. As a user process executes, the bit associated with each page referenced is set (to 1) by the hardware. After some time, we can determine which pages have been used and which have not been used by examining the reference bits. We do not know the *order* of use, but we know which pages were used and which were not used. This partial ordering information leads to many page-replacement algorithms that approximate LRU replacement.

7. Explain Counting-Based Page Replacement algorithm.

. Counting-Based Page Replacement

There are many other algorithms that can be used for page replacement. For example, we could keep a counter of the number of references that have been made to each page, and develop the following two schemes.

1. The least frequently used (LFU) page-replacement algorithm requires that the page with the smallest count be replaced. The reason for this selection is that an actively used page should have a large reference count. This algorithm suffers from the situation in which a page is used heavily during the initial phase of a process, but then is never used again. Since it was used heavily, it has a large count and remains in memory even though it is no longer needed.

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

8. Explain Global Versus Local Allocation of frames.

Global Versus Local Allocation

Another important factor in the way frames are allocated to the various processes is page replacement. With multiple processes competing for frames, we can classify page-replacement algorithms into two broad categories: **global replacement and local replacement**. **Global replacement allows a process to select a replacement frame from the set of all frames, even if that frame is currently allocated to some other process; one process can take a frame from another. Local replacement requires that each process select from only its own set of allocated frames.**

For example, consider an allocation scheme where we allow high-priority processes to select frames from low-priority processes for replacement. A process can select a replacement from among its own frames or the frames of any lower-priority process. This approach allows a high-priority process to increase its frame allocation at theexpense of the low-priority process.

With a local replacement strategy, the number of frames allocated to a process does not change. With global replacement, a process may happen to select only frames allocated to other processes, thus increasing the number of frames allocated to it (assuming that other processes do not choose *its* frames for replacement)

One Marks Questions.

1. What is a File?
   A file is a named collection of related information that is recorded on secondary storage

2. What is a File pointer?
   If the operating system does not include file offset along with the read and write system calls, then the OS should kept track of the last read/write locations as a current file position pointer, known as file pointer.

3. What is a directory?

4. What is a Free-space list?
   The free-space list records all *free* disk blocks-those not allocated to some file or directory.

5. What is meant by Counting?

6. What is meant by Grouping?

Five Marks Questions.

1. What is a File? Explain the operations that can be performed on a file.
   A file is a named collection of related information that is recorded on secondary storage

File Operations

1. 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.
- The directory entry records the name of the file and the location in the file system, and possibly other information.

2. 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. Given .the name of the file, the system searches the directory to find the location of the file. The system must keep a *write* pointer to the location in the file where the next write is to take place. The write pointer must be updated whenever a write occurs.

3. 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. Again, the directory is searched for the associated directory entry, and the system needs to keep a *read* pointer to the location in the file where the next read is to take place. Once the read has taken place, the read pointer is updated. A given process is usually only reading or writing a given file and the current operation location is kept as a per-process current-

file-position pointer. Both the read and write operations use this same pointer, saving space and reducing the system complexity.

4. **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.

5. **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.

6. **Repositioning within a file:** The directory is searched for the appropriate entry and the current-file position is set to a given value.

- **Append:** Appending new information to the end of an existing file.
- **Renaming:** Remaining an existing file's file name.
- **List:** Output the contents of a file.
- **Edit:** Edit or modify the contents of a existing file**.**
- **Copy:** creating a new file, and reading from the old and writing to the new.
- **Get and Set:** This operations that allow a user to get and set the various attributes of a file.

2. Explain Two-level Directory structure of directory.

2. Two-Level Directory

A single-level directory often leads to confusion of file names between different users. The standard solution is to create a *separate* directory for each user.

In the two-level directory structure, each user has own **user file directory (UFD).** Each UFD has a similar structure, but lists only the files of a single user. When a user job starts or a user logs in, the system's **master file directory (MFD)** is searched. The MFD is indexed by user name or account number, and each entry points to the UFO for that user (Figure).

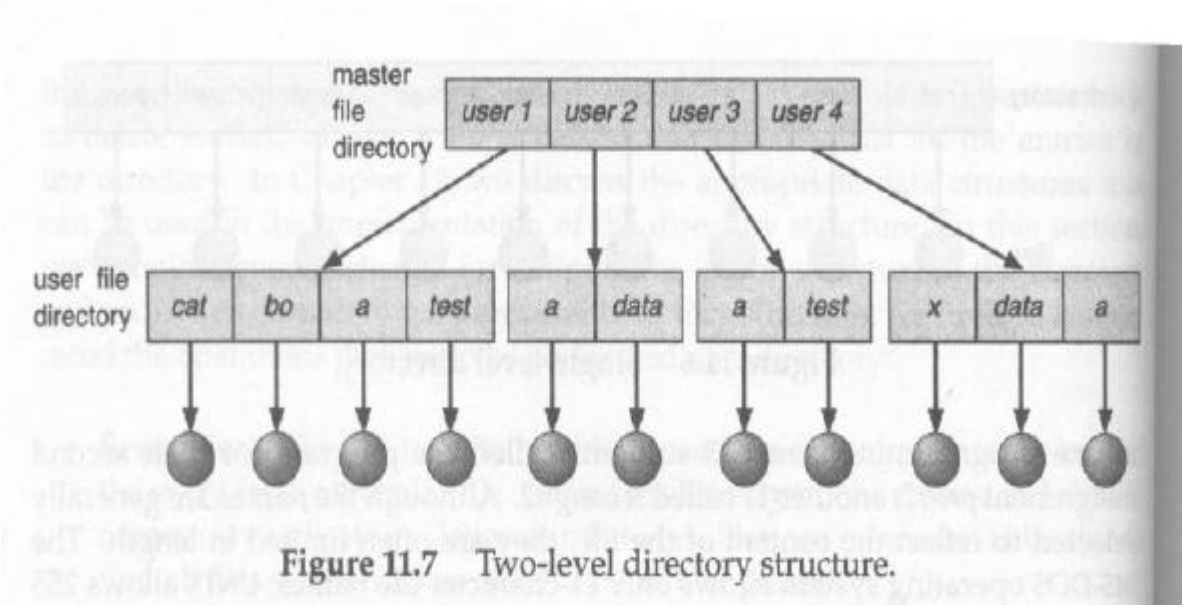


Figure 11.7 Two-level directory structure.

When a user refers to a particular file, only his own UFD is searched. Thus, different users may have files with the same name, as long as all the file names within each UFD are unique.

To create a file for a user, the operating system searches only that user's UFD to ascertain whether another file of that name exists. To delete a file, the operating system confines its search to the local UFD; thus, it cannot accidentally delete another user's file that has the same name.

The user directories themselves must be created and deleted as necessary. A special system program is run with the appropriate user name and account information. The program creates a new UFO and adds an entry for it to the MFD. The execution of this program might be restricted to system administrators

Although the two-level directory structure solves the name-collision problem, it still has disadvantages. This structure effectively isolates one user from another. This isolation is an advantage when the users are completely independent, but is a disadvantage when the users *want* to cooperate on some task and to access one another's files. Some systems simply do not allow local user files to be accessed by other users.

If access is to be permitted, one user must have the ability to name a file in another user's directory. To name a particular file uniquely in a two-level directory, we must give both the user name and the file name. A two-level directory can be thought of as a tree, or at least an inverted tree, of height 2. The root of the tree is the MFD. Its direct descendants are the UFDs. The descendants of the UFDs are the files themselves. The files are the leaves of the tree. Specifying a user name and a file name defines a path in the tree from the root (the MFD) to a leaf (the specified file). Thus, a user name and a file name define a *path name*. Every file in the system has a path name. To name a file uniquely, a user must know the path name of the file desired.

3. Explain Single level Directory structure of directory.

The simplest directory structure is the single-level directory. All files at contained in the same directory, which is easy to support and understand (Figure).

A single-level directory has significant limitations, however, when the' number of files increases or when the system has more than one user. Since, all files are in the same directory, they must have unique names. If two user call their data file *test,* then the unique-name rule is violated. For example in one programming class, 23 students called the program for their second assignment *prog2;* another 11 called it *assign2.* Although file names are generally selected to reflect the content of the file, they are often limited in length. The MS-DOS operating system allows only 11-character file names; UNIX allows 255 characters.

- 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.
- It is not uncommon for a user to have hundreds of files on one computer system and an equal number of additional files on another system.

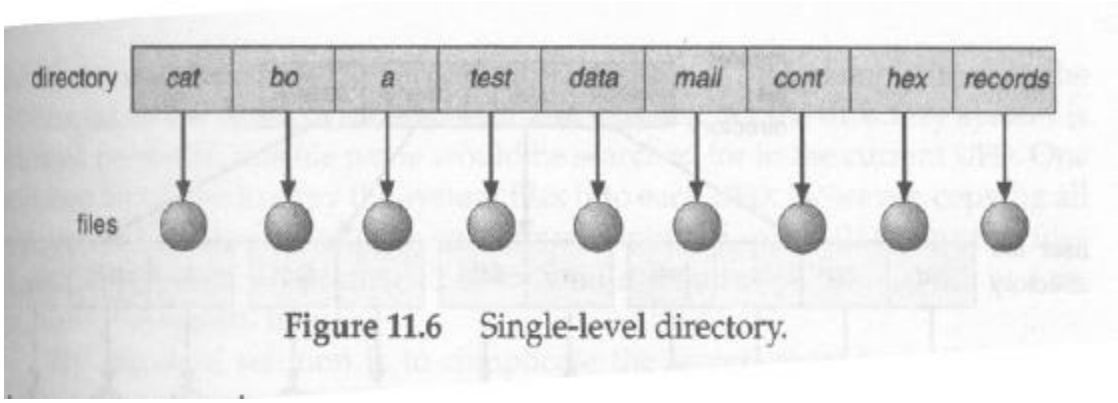


Figure 11.6 Single-level directory.

4. Explain Tree Structure Directory structure of directory.

Tree-Structured Directories

Once we have seen how to view a two-level directory as a two-level tree, the *natural* generalization is to extend the directory structure to a tree of arbitrary height (Figure). **This generalization allows users to create their own sub directories and to organize their files accordingly every file in the system has a unique path name. A path name is the path from the root, through all the subdirectories, to a specified file.**

A directory (or subdirectory) contains a set of files or subdirectories. A directory is simply another file, but it is treated in a special way. All directories have the same internal format. One bit in each directory entry defines the entry as a file (0) or as a subdirectory (1). Special system calls are used to create and delete directories.

The initial current directory of a user is designated when the user job starts .r the user logs in. The operating system searches the accounting file to find an entry for this user (for accounting

purposes). In the accounting file is a pointer to (or the name of) the user's initial directory. This pointer is copied to a local variable for this user that specifies the user's initial current directory.
Path names can be of two types: ***absolute* path names or *relative* path names**. An absolute path name begins at the root and follows a path down to the specified file, giving the directory names on the path. A relative path name defines a path from the current directory. For example, in the tree-structured file system of Figure

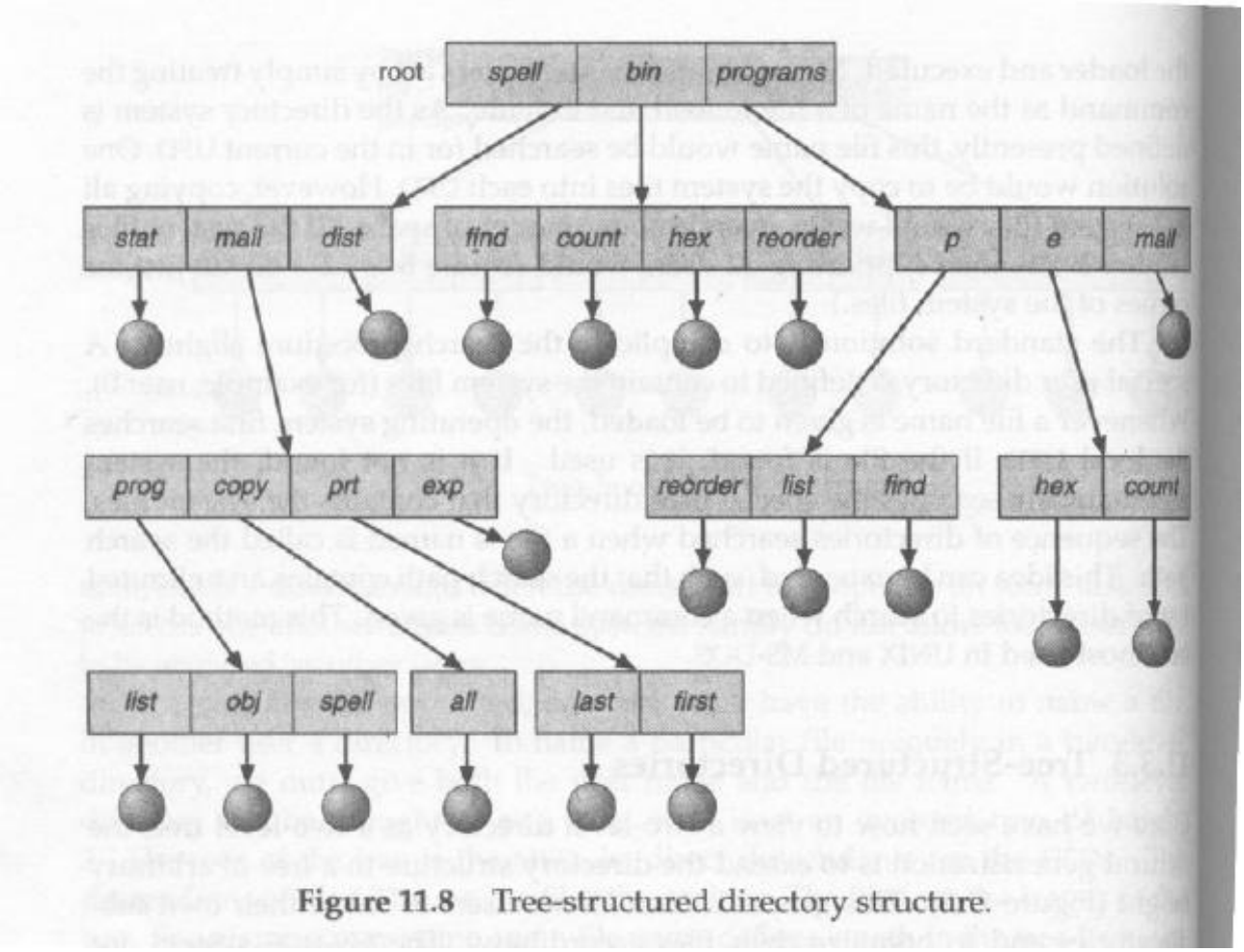


Figure 11.8 Tree-structured directory structure.

5. Explain the different file types and file operations.

File Operations

1. 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.
- The directory entry records the name of the file and the location in the file system, and possibly other information.

2. 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. Given .the name of the file, the system searches the directory to find the location of the file. The system must keep a *write* pointer to the location in the file where the next write is to take place. The write pointer must be updated whenever a write occurs.

3. 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. Again, the directory is searched for the associated directory entry, and the system needs to keep a *read* pointer to the location in the file where the next read is to take place. Once the read has taken place, the read pointer is updated. A given process is usually only reading or writing a given file and the current operation location is kept as a per-process current-file-position pointer. Both the read and write operations use this same pointer, saving space and reducing the system complexity.

4. 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.

5. 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.

6. Repositioning within a file: The directory is searched for the appropriate entry and the current-file position is set to a given value.

- **Append:** Appending new information to the end of an existing file.
- **Renaming:** Remaining an existing file's file name.
- **List:** Output the contents of a file.
- **Edit:** Edit or modify the contents of a existing file**.**
- **Copy:** creating a new file, and reading from the old and writing to the new.
- **Get and Set:** This operations that allow a user to get and set the various attributes of a file.

File Types

File type	Usual extension	Function
executable	exe, com, bin	**read to run machine-language program**
Object	obj,o	**compiled, machine language, not linked**
Source code	c, cc, java, pas, asm	**source code in various languages**
Batch	bat, sh	**commands to the command interpreter**
Text word processor	txt, doc ,wp , rtf	**textual data, documents various word-processor**
Library	lib	**libraries of routines for programmers**
Image	bmp, jpg	**Different form of image file**
Data base file	Dbf mdb	**Different form of DBMS file**
Help file	Hlp	**Application help file**
Sound file	Au , m3u	**Sound files**
Movie clip	Avi miv	**Different application movie format file**

6. Explain sequential and direct file access methods

1 Sequential Access

The simplest access method is sequential access. Information in the file is processed in order, one record after the other. This mode of access is by far the most common; for example, editors and compilers usually access files in this fashion. The bulk of the operations on a file is reads and writes. A read operation reads the next portion of the file and automatically advances a file pointer, which tracks the I/O location. Similarly, a write appends to the end of the file and advances to the end of the newly written material (the new end of file)! Such a file can be reset to the beginning and, on some systems, a program maybe able to skip forward or backward *n* records, for some integer n-perhaps: only for *n* =1. Sequential access is depicted in Figure Sequential access based on a tape model of a file, and works as well on sequential-access devices; as it does on random-access ones.

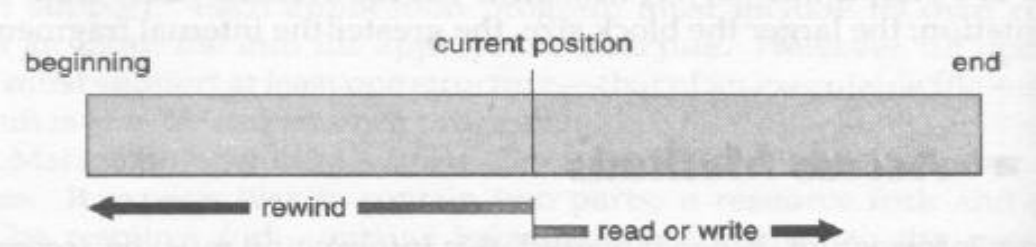


Figure 11.2 Sequential-access file.

Direct Accesses

Another method is direct access (or relative access). A file is made up of fixed length logical records that allow programs to read and write records rapidly in no particular order. The direct-access method is based on a disk model a file, since disks allow random access to any file block. For direct access, in file is viewed as a numbered sequence of blocks or records. A direct-access allows arbitrary blocks to be read or written. Thus, we may read block 14, then read block 53, and then write block 7. There are no restrictions on the order reading or writing for a direct-access file.

Direct-access files are of great use for immediate access to large amount of information. Databases are often of this type. When a query concerning particular subject arrives, we compute which block contains the answer, and then read that block directly to provide the desired information.

As a simple example, on an airline-reservation system, we might store the information about a particular flight (for example, flight 713) in the block identified by the flight number. Thus, the number of available seats for flight 713 is stored in block 713 of the reservation file. To store information about a larger set, such as people, we might compute a hash function on the people's names, or search a small in-memory index to determine a block to read and search.

sequential access	implementation for direct access
reset	*cp = 0;*
read next	*read cp;* *cp = cp+1;*
write next	*write cp;* *cp = cp+1;*

.

7. Explain any four operations that can be performed on a directory.

Operation performed on directory

- **Search for a file**: We need to be able to search a directory structure to find the entry for a particular file. Since files have symbolic names and similar names may indicate a relationship between files, we may want to be able to find all files whose names match a particular pattern.
- **Create a file**: New files need to be created and added to the directory.
- **Delete a file**: When a file is no longer needed, we want to remove it from the directory.

- **List a directory:** We need to be able to list the files in a directory, and the contents of the directory entry for each file in the list.

8. Explain Free-Space Management.

Free-Space Management

Since disk space is limited, we need to reuse the space from deleted files for new files, if possible. To keep track of free disk space, the system maintains a free-space list. The free-space list records all *free* disk blocks-those not allocated to some file or directory. **To create a file, we search the free-space list for the required amount 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.

9. Write a note on following.

a). Bit Vectors.

Frequently, the free-space list is implemented as a bit map or bit vector. Each block is represented by 1 bit. If the block is free, the bit is 1; if the block is allocated, the bit is O.

For example, consider a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 are free, and the rest of the blocks are allocated. The free space bit map would be

0011110011111100011000000011100000 ...

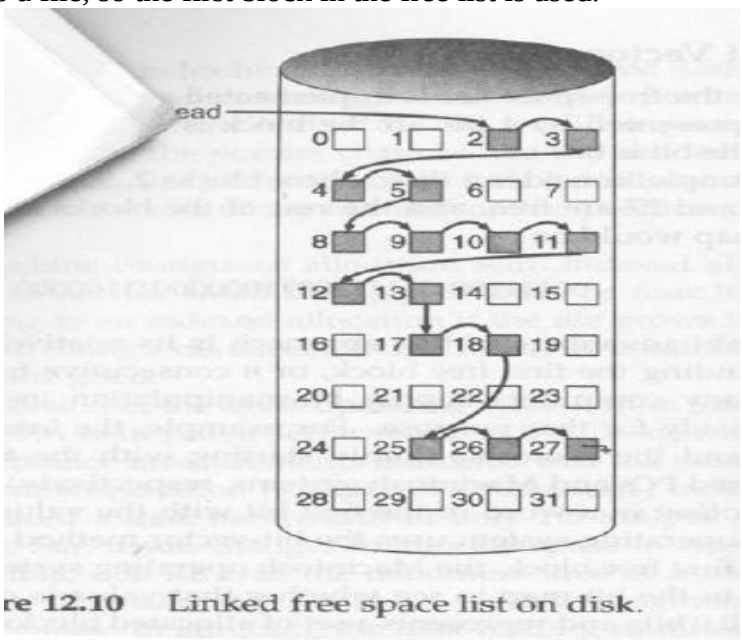
The main advantage of this approach is it's relatively simplicity and efficiency in finding the first free block, or *n* consecutive free blocks on the disk. Indeed, many computers supply bit-manipulation instructions that can be used effectively for that purpose Location of the first free block.

b). Linked list.

Linked Lists

Another approach to free-space management is to link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching

it in memory. This first block contains a pointer to the next free disk block, and so on. We would keep a pointer to block 2, as the first free block. Block 2 would contain a pointer to block 3, which would point to block 4, which would point to block 5, which would point to block 8, and so on (Figure). However, this scheme is not efficient; to traverse the list, we must read each block, which requires substantial I/O time. Fortunately, traversing the free list is not a frequent action. Usually, the operating system simply needs a free block so that it can allocate that block to a file, so the first block in the free list is used.



re 12.10 Linked free space list on disk.

c). Grouping.

A modification of the free-list approach is to store the addresses of *n* free blocks in the first free block. The first *n—1* of these blocks are actually free. The last block contains the addresses of another *n* free blocks, and so on. The importance of this implementation is that the addresses of a large number of free blocks can be found quickly; unlike in the standard linked-list approach.

d).Counting.

Counting

Another approach is to take advantage of the fact that, generally, several contiguous blocks may be allocated or freed simultaneously, particularly when space is allocated with the contiguous-allocation algorithm or through clustering**. Thus, rather than keeping a list of *n* free disk addresses, we can keep the address of the first free block and the number *n* of free contiguous blocks that follow the first block. Each entry in the free-space list then consists of a disk address and a count.** Although each entry requires more space than would a simple disk address, the overall list will be shorter, as long as the count is generally greater than 1.

One Marks Questions

1. The time for the disk arm to move the heads to the cylinder containing the desired sector is called…… **seek time** ……..

2. Define Rotation Latency

The rotational latency is the additional time waiting for the disk to rotate the desired sector to the disk head.

3. What is disk bandwidth?

The disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.

4. What is a Seek time?

The seek time is the time for the disk arm to move the heads to the cylinder containing the desired sector.

5. What are the two major components of the access time?

1.seek time

2. **rotational latency**

6. What are the information any disk scheduling request specifies?

The request specifies several pieces of information:

- **Whether this operation is input or output**
- **What the disk address for the transfer is**
- **What the memory address for the transfer is.**
- **What the number of bytes to be transferred is**

7. What is FCFS disk scheduling?

8. What is SSTF disk scheduling?

9. What is SCAN disk scheduling?

In the SCAN algorithm, the disk arm starts at one end of the disk, and moves toward the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk. At the other end, the direction of head movement is reversed, and servicing continues. The head continuously scans back and forth across the disk.

10. What is C-SCAN disk scheduling?

C-SCAN Scheduling

Circular SCAN (C-SCAN) scheduling is a variant of SCAN designed to provide a more uniform wait time. Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way. **When the head reaches the other end, however, it immediately returns to the beginning of the disk. Without servicing any requests on the return trip.**

11. What is LOOK disk scheduling?

12. What is C-LOOK disk scheduling?
13. What is the draw-back of SSTF disk scheduling?

14. The bootstrap is stored in…… **read-only memory (ROM).**…….

Five Marks Questions

1. Explain following Disk scheduling with example.
   a) FCFS.:

FCFS Scheduling

The simplest form of disk scheduling is, of course, the **first-come, first-served (FCFS) algorithm.** This algorithm is intrinsically fair, but it generally does not provide the fastest service. Consider, for example, a disk queue with request I/O to blocks on cylinders

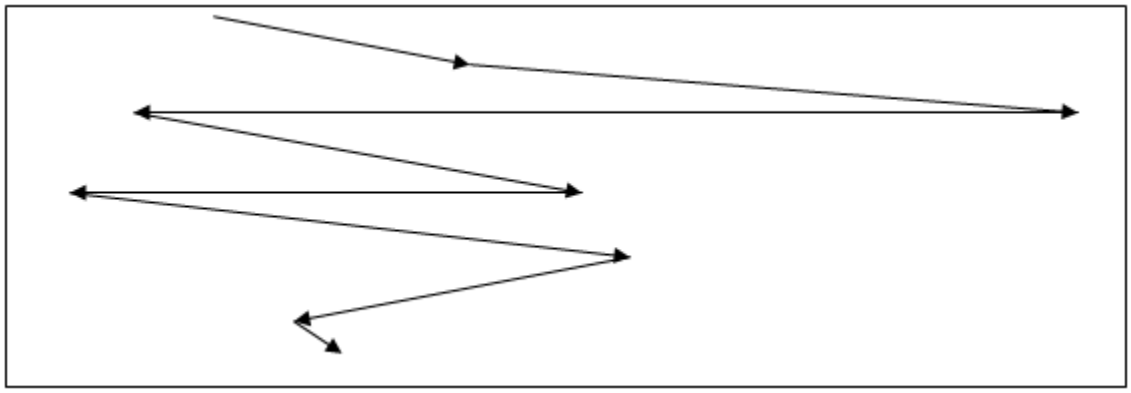
98, 183, 37,122, 14, 124, 65, 67,

in that order. If the disk head is initially at cylinder 53, it will first move from 53 to 98, then to 183, 37, 122, 14, 124, 65, and finally to 67, for a total head movement of 640 cylinders. This schedule is diagrammed in Figure.

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

Head starts at 53

0 14 37 53 65 67 98 122 124 183 199



b) SSTF.

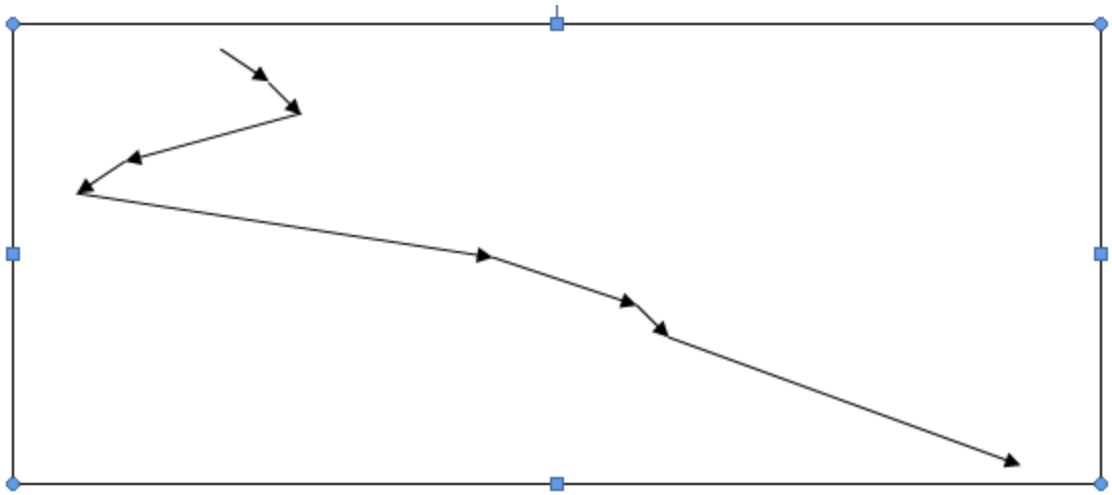
SSTF Scheduling

It seems reasonable to service all the requests close to the current head position, before moving the head far away to service other requests. This assumption is the basis for the **shortest-seek-time-first** (SSTF) algorithm. **The SSTF algorithm selects the request with the minimum seek time from the current head position.** Since seek time increases with the number of cylinders traversed by the head**. SSTF chooses the pending request closest to current head position.**

For our example request queue, the closest request to the initial head position (53) is at cylinder 65. Once we are at cylinder 65, the next closest request is at cylinder 67. From there, the request at cylinder 37 is closer than 98, so 37 is served next. Continuing, we service the request at cylinder 14, then 98, 122, 124, and finally 183 (Figure). This scheduling method results in a total head movement of only 236 cylinders-little more than one-third of the distance needed for FCFS scheduling of this request queue. This algorithm gives a substantial improvement in performance.

Queue = 98, 183, 37, 122, 14, 124, 65, 67
Head starts at 53

0 14 37 53 65 67 98 122 124 183 199



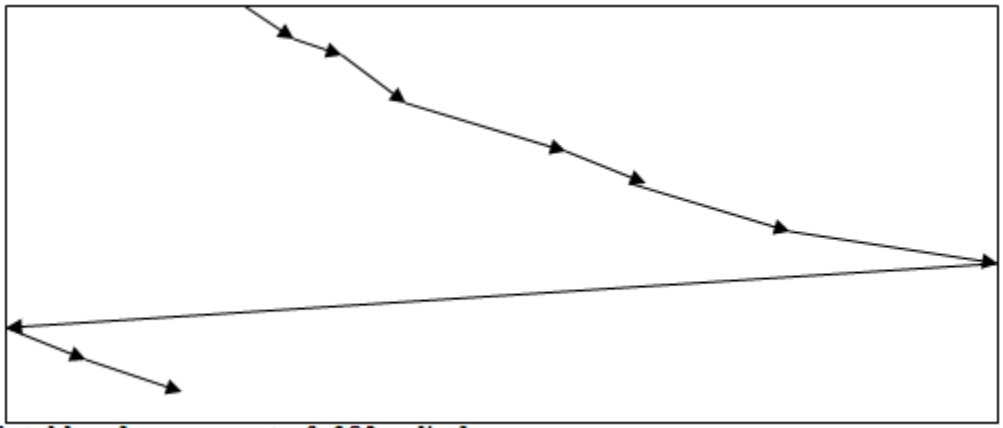
c) C-SCAN

C-SCAN Scheduling

Circular SCAN (C-SCAN) scheduling is a variant of SCAN designed to provide a more uniform wait time. Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way. **When the head reaches the other end, however, it immediately returns to the beginning of the disk. Without servicing any requests on the return trip (Figure).** The C-SCAN scheduling algorithm essentially treats the cylinders as a circular list that wraps around from the final cylinder to the first one.

Queue = 98, 183, 37, 122, 14, 124, 65, 67
Head starts at 53

0 14 37 53 65 67 98 122 124 183 199



Total head movement of 382 cylinders

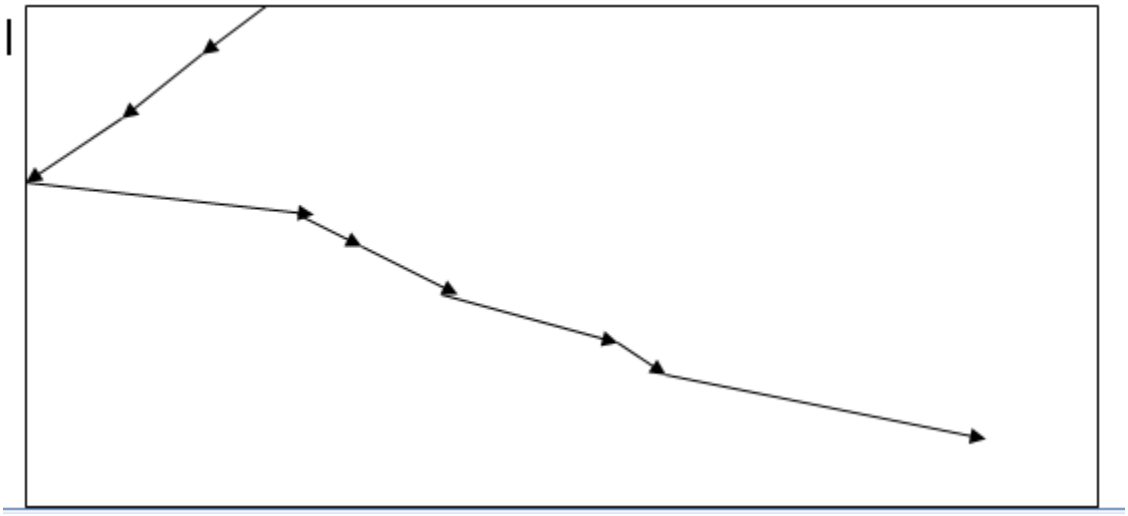
d) SCAN

In the SCAN algorithm, the disk arm starts at one end of the disk, and moves toward the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk. At the other end, the direction of head movement is reversed, and servicing continues. The head continuously scans back and forth across the disk.

Before applying SCAN to schedule the requests on cylinders *98* 183, 37, 122, 14, 124, 65, and 67, we need to know the direction of head movement, in addition to the head's current position (53). If the disk arm is moving toward 0, the head will service 37, and then 14. At cylinder 0, the arm will reverse and will move toward the other end of the disk, servicing the requests at 65, 67, 98, 122, 124, and 183 (Figure). -If a request arrives in the queue just in front of the head, it will be serviced almost immediately; a request arriving just behind the head will have to wait until the arm moves to the end of the disk, reverses direction, and comes back.

Queue = 98, 183, 37, 122, 14, 124, 65, 67
Head starts at 53

0 14 37 53 65 67 98 122 124 183 199



e) LOOK

f) C-LOOK.

2. Explain the concept of Boot block and Bad block.

Boot Block

For a computer to start running-for instance, when it is powered up or rebooted-it needs to have an initial program to run. This initial bootstrap program tends to be simple. It initializes all aspects of the system, from CPU registers to device controllers and the contents of main memory, and then starts the operating system. To do its job, the bootstrap program finds the operating system kernel on disk, loads that kernel into memory, and jumps to an initial address to begin the operating-system execution.

For most computers, the bootstrap is stored in read-only memory (ROM).

This location is convenient, because ROM needs no initialization and is at a fixed location that the processor can start executing when powered up or reset. And, since ROM is read only, it cannot be infected by a computer virus. The problem is that changing this bootstrap code requires changing the ROM hardware chips. For this reason, most systems store a tiny bootstrap loader program in the boot ROM, whose only job is to bring in a full bootstrap program from disk. The full bootstrap program can be changed easily: A new version is simply written onto the disk. (The full bootstrap program is stored in a partition called the boot blocks, at a fixed location on the disk. A disk that has a boot partition is called a boot disk or system disk.)

Bad Blocks

Because disks have moving parts and small tolerances (recall that the disk head flies just above the disk surface), they are prone to failure. Sometimes the failure is complete, and the disk needs to be replaced, and its contents restored from backup media to the new disk. More frequently, one or more sectors become defective. Most disks even come from the factory with bad blocks. Depending on the disk and controller in use, these blocks are handled in a variety of ways.

For instance, the MS-DOS format command does a logical format and, as a part of the process, scans the disk to find bad blocks. If format finds a bad block, it writes a special value into the corresponding FAT entry to tell the allocation routines not to use that block. If blocks go bad during normal operation, a special program (such as chkdsk) must be run manually to search for the bad blocks and to lock them away as before. Data that resided on the bad blocks usually are lost.

3. Write a note on SSTF disk scheduling. What is the problem in this scheduling algorithm?

SSTF Scheduling

It seems reasonable to service all the requests close to the current head position, before moving the head far away to service other requests. This assumption is the basis for the **shortest-seek-time-first** (SSTF) algorithm. **The SSTF algorithm selects the request with the minimum seek time from the current head position.** Since seek time increases with the number of cylinders traversed by the head**. SSTF chooses the pending request closest to current head position**

4. Write a note on Disk Structure.

Disk Structure

Disks provide the bulk of secondary storage for modem computer systems. Magnetic tape was used as an early secondary-storage medium, but the access time is much slower than for disks. Thus, tapes are currently used mainly for backup, for storage of infrequently used information, as a medium for transferring information from one system to another, and for storing quantities of data so large that they are impractical as disk systems

Modern disk drives are addressed as large one-dimensional arrays of logical blocks, where the logical block is the smallest unit of transfer. The size of a logical block is usually 512 bytes, although some disks can be low-level formatted to choose a different logical block size, such as 1,024 bytes.

The one-dimensional array of logical blocks is mapped onto the sectors of the disk sequentially. Sector 0 is the first sector of the first track on the outermost cylinder. The mapping proceeds in order through that track, then through the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost.

By using this mapping, we can-at least in theory-convert a logical block number into an old-style disk address that consists of a cylinder number, a track number within that cylinder, and a sector number within that track.

The number of sectors per track has been increasing as disk technology improves, and the outer zone of a disk usually has several hundred sectors per track. Similarly, the number of cylinders per disk has been increasing; large disks have tens of thousands of cylinders

5. Write a note on Disk formatting.

Disk Formatting

A new magnetic disk is a blank slate. It is just platters of a magnetic recording material. Before a disk can store data, it must be divided into sectors that the disk controller can read and write. This process is called low-level formatting (or physical formatting). Low-level formatting fills the disk with a special data structure for each sector. The data structure for a sector typically

consists of a header, a data area (usually 512 bytes in size).

Most hard disks are low-level formatted at the factory as a part of the manufacturing process. This formatting enables the manufacturer to test the disk and to initialize the mapping from logical block numbers to defect-free sectors on the disk. For many hard disks, when the disk controller is instructed to low-level format the disk, it can also be told how many bytes of data space to leave between the header and trailer of all sectors. It is usually possible to choose among a few sizes, such as 256, 512, and 1,024 bytes.

To use a disk to hold files, the operating system still needs to record its own data structures on the disk. It does so in two steps. The first step is to partition the disk into one or more groups of cylinders. The operating system can treat each partition as though it were a separate disk. For instance, one partition can hold a copy of the operating system's executable code, while another holds user files. After partitioning, the second step is logical formatting (or creation of a file system). In this step, the operating system stores the initial file-system data structures onto the disk. These data structures may include maps of free and allocated space (a FAT or Inodes) and an initial empty directory.

Problems.

1. Calculate the total head movement using
   a). SCAN scheduling.
   b). C-SCAN scheduling.
   Queue=88, 180, 34,119,10,120,68,69.
   Head Starts at 50.

2. Suppose that a disk drive has 5,000 cylinders, numbered 0 to 4999. The drive is currently serving a request at cylinder 143, and previous request was at cylinder 125.The queue of pending requests in FIFO order is

   86, 1470, 913,1774,948,1509,1022,1750,130.

   Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests for each of the disk scheduling algorithms?

a). FCFS.
b). SSTF.
c). SCAN.
d).C-SCAN.
e). LOOK.
f).C-LOOK.