TED (10)-3071 Reg. No. …….

(REVISION-2010) Signature ……..……

FORTH SEMESTER DIPLOMA EXAMINATION IN ENGINEERING/ TECHNOLIGY- MARCH, 2014

OPERATING SYSTEM

(Common to CT, CM and IF)

[*Time:* 3 hours

(Maximum marks: 100)

Marks

PART –A
(Maximum marks: 10)

I. Answer all questions in a sentence

1. Write the major functions of Operating systems.

   - Execute user programs and make solving user problems easier.
   - Make the computer system convenient to use

2. What is thread?

   A thread is a basic unit of CPU utilization, consisting of a program counter, a stack, and a set of registers, (and a thread ID.) Traditional (heavyweight) processes have a single thread of control - There is one program counter, and one sequence of instructions that can be carried out at any given time

3. What is 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 memory chunks of varying sizes onto the backing store; most memory-management schemes used before the introduction of paging suffered from this problem.

4. List different disk scheduling algorithms

   - FCFS

- SSTF
- SCAN
- CSCAN

5. List the file attributes

    - Name
    - Identifier
    - Type
    - Location
    - Size
    - Protection
    - Time, date, and user identification

PART – B

II. Answer *any five* questions. Each question carries 6 marks

1. Compare the time sharing system and real-time system.

    Time sharing system

    Timesharing refers to the allocation of computer resources in a time-dependent fashion to several programs simultaneously.

    - Time sharing, or multitasking, is a logical extension of multiprogramming.
    - Multiple jobs are executed simultaneously by switching the CPU between them.
    - Each job gets a predetermined "time slice"
    - Time slice is defined by the OS, for sharing CPU time between processes.
    - At the end of time slice current job is set aside and a new one starts
    - In this, the CPU time is shared by different processes, so it is called as "Time sharing Systems".
    - Examples: Multics, Unix, etc.,

    Real-time system

    Real time system is an on-line processing system. It receives and processes data quickly enough to produce output to control the outcome of the ongoing activity.

- It has well defined and fixed time constraints.
- Processing must be done within the defined constraints, or the system will fail.
- Real time systems may be either hard or soft real time
    - **Hard real time s/m**: guarantees that the critical tasks be completed on time.
    - **Soft real time s/m**: less restrictive system where a critical real-time task gets priority over other tasks and retains that priority until it completes.

2. Explain multiprocessor system

   Although single-processor systems are most common, **multiprocessor systems** (also known as **parallel systems** or **tightly coupled systems)** are growing in importance. Such systems have two or more processors in close communication, sharing the computer bus and sometimes the clock, memory, and peripheral devices.

   Multiprocessor systems have three main advantages

   1. Increased throughput. By increasing the number of processors, we expect to get more work done in less time. The speed-up ratio with N processors is not N, however; 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 cost less than equivalent multiple single-processor systems, because they can share peripherals, mass storage, and power supplies.
   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.

3. What is PCB? Explain it.

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

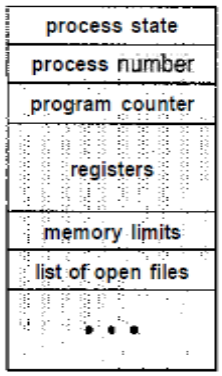
   - **Process state.** The state may be new, ready, running, and waiting, halted, and so on.
   - **Program counter.** The counter indicates the address of the next instruction to be executed for this process.

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

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

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



Process control block (PCB).

4. What is deadlock? What are its causes?

   Deadlock is a situation which occurs when a process enters a waiting state because a resource requested by it is being held by another waiting process, which in turn is waiting for another resource.

   Necessary conditions for deadlock to occur are:

   - Mutual exclusion: At least one resource must be held in a non-sharable mode; 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.
   - 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.

- No pre-emption: Resources cannot be pre-empted; that is, No resource can be forcibly removed from a process holding it.
- Circular wait: A set{P0, P1,....., Pn) of waiting processes must exist such that P0 is waiting for a resource that is held by P1, P1 is waiting for a resource that is held by P2, ...,Pn-1 is waiting for a resource that is held by Pn and Pn is waiting for a resource that is held by P1.

All four conditions must hold simultaneously for a deadlock to occur and the conditions are not completely independent

5. What is fragmentation? Differentiate between internal and external fragmentation.

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

   External Fragmentation – total memory space exists to satisfy a request, but it is not contiguous

   - 50-percent rule: for first fit, given N allocated blocks, another 0.5N will be lost to fragmentation
     - 1/3 of memory may be unusable!!!

   Internal Fragmentation – allocated memory may be slightly larger than requested memory; the extra bytes can-not be used by other processes

6. Write and explain various file operations

   - **Creating a file.** Two steps are necessary to create a file. First, space in the file system must be found for the file. We discuss how to allocate space for the file in Chapter 11. Second, an entry for the new file must be made in the directory.

   - **Writing a file.** To write a file, we make a system call specifying both the name of the file and the information to be written to the file. Given the name of the file, the system searches the directory to find the file's location. 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.

- **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 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.

- **Repositioning within** a **file.** The directory is searched for the appropriate entry, and the current-file-position pointer is repositioned to a given value. Repositioning within a file need not involve any actual I/O. This file operation is also known as a file *seeks.*

- **Deleting a file.** To delete a file, we search the directory for the named file. Having found the associated directory entry, we release all file space, so that it can be reused by other files, and erase the directory entry.

- **Truncating a file.** The user may want to erase the contents of a file but keep its attributes. Rather than forcing the user to delete the file and then recreate it, this function allows all attributes to remain unchanged—except for file length—but lets the tile be reset to length zero and its file space released.

7. Explain swap space management

   Swap-space management is another low-level task of the operating system. Virtual memory uses disk space as an extension of main memory. Since disk access is much slower than memory access, using swap space significantly decreases system performance. The main goal for the design and implementation of swap space is to provide the best throughput for the virtual memory system

   Swap-Space Use

   Swap space is used in various ways by different operating systems, depending on the memory-management algorithms in use. For instance, systems that implement swapping may use swap space to hold an entire process image, including the code and data segments. Paging systems may simply store pages that have been pushed out of main memory.

   Swap-Space Location

A swap space can reside in one of two places: It can be carved out of the normal file system, or it can be in a separate disk partition. If the swap space is simply a large file within the file system, normal file-system routines can be used to create it, name it, and allocate its space. This approach, though easy to implement, is inefficient. Navigating the directory structure and the disk-allocation data structures takes time and (potentially) extra disk accesses

PART – C

(Answer *one* full question from each unit. Each question carries 15 marks)

UNIT – I

III.

a) Explain functions of assembler and loader 8

Assembler

- Convert mnemonic operation codes to machine language equivalents
- Convert symbolic operands to machine addresses (pass 1)
- Build machine instructions
- Convert data constants to internal representations

Loader

- Loading – brings the object program into memory for execution
- Allocation - The loader determines and allocates the required memory space for the program to execute properly.
- Relocation – modifies the object program so that it can be loaded at an address different from the location originally specified
- Linking – combines two or more separate object programs and also supplies the information needed to reference them

b) Compare networking system and protection system 7

Networking system

- A distributed system is a collection processor that does not share memory or a clock. Each processor has its own local memory.

- The processors in the system are connected through a communication network.
- Communication takes place using a protocol.
- A distributed system provides user access to various system resources.
- Access to a shared resource allows:
    - Computation speed-up
    - Increased data availability
    - Enhanced reliability

Protection system

- *Protection* refers to a mechanism for controlling access by programs, processes, or users to both system and user resources.
- The protection mechanism must:
    - Distinguish between authorized and unauthorized usage.
    - Specify the controls to be imposed.
    - Provide a means of enforcement.

OR

IV.

a) Discuss the following in detailed manner

i. Batch system

ii. Multiprogramming system

iii. Functions of compiler 9

i. **Batch processing system**: In a batch system, more processes are submitted than can be executed immediately. These processes are spooled to a mass-storage device (typically a disk), where they are kept for later execution. Finally, the operator retrieves the output of all these jobs and returns them to the concerned users. Batch processing is most suitable for tasks where a large amount of data has to be processed on a regular basis

ii. **Multiprogramming system:** Multiprogramming increases CPU utilization by organizing jobs (code and data) so that the CPU always has one to execute In a multi-

programmed 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

iii. A **compiler** is responsible for translating a source code, which is written in a programming language, into a target language. This is most commonly done in order to create an executable program. A compiler is mainly used for programs that translate a source code into an assembly language or machine code, which are both a lower level language than the source code

b) Explain the following

i. Process management

ii. File management 6

Process Management

A *process* is a program in execution. A process needs certain resources, including CPU time, memory, files, and I/O devices, to accomplish its task.

- The operating system is responsible for the following activities in connection with process management.
- Process creation and deletion.
- Process suspension and resumption.
- Provision of mechanisms for:
  - Process synchronization
  - Process communication

File Management

- A file is a collection of related information defined by its creator. Commonly, files represent programs (both source and object forms) and data.
- The operating system is responsible for the following activities in connections with file management:

  – File creation and deletion.

  – Directory creation and deletion.

– Support of primitives for manipulating files and directories.

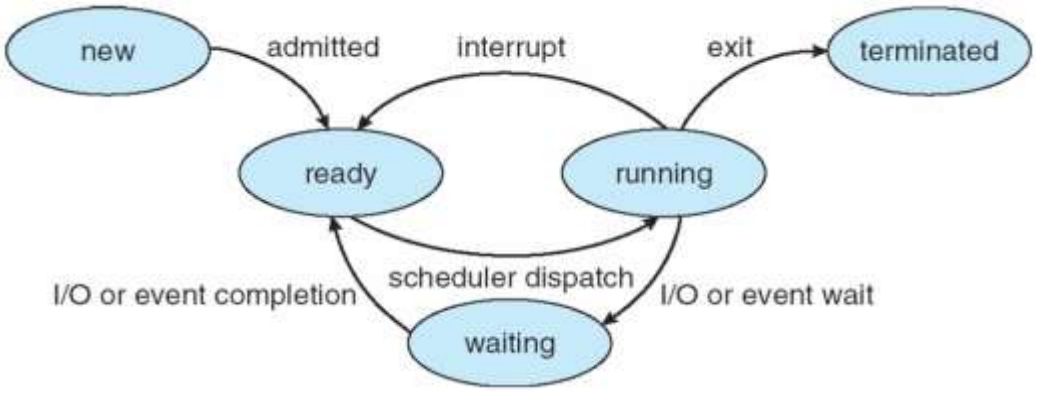
– Mapping files onto secondary storage.

– File backup on stable (nonvolatile) storage media

UNIT – II

V.

a) Explain different process state with diagram 7



As a process executes, it changes state

- **new**: The process is being created
- **running**: Instructions are being executed
- **Waiting**: The process cannot run at the moment, because it is waiting for some resource to become available or for some event to occur. For example the process may be waiting for keyboard input, disk access request, inter-process messages, a timer to go off, or a child process to finish.
- **ready**: The process has all the resources available that it needs to run, but the CPU is not currently working on this process's instructions
- **terminated**: The process has finished execution

b) Define critical section problem. Explain its solutions 8

Consider a system consisting of *n* processes $\{P_1, P_2, ..., P_{n-1}\}$. Each process has a segment of code, called a critical section, in which the process maybe changing common variables, updating a table, writing a file, and so on. 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.

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

1. **Mutual exclusion.** If process P; is executing in its critical section, then no other processes can be executing in their critical sections.
2. **Progress.** If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder sections can participate in the decision on which will enter its critical section next, and this selection cannot be postponed indefinitely.
3. **Bounded waiting.** There exists a bound, or limit, 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.

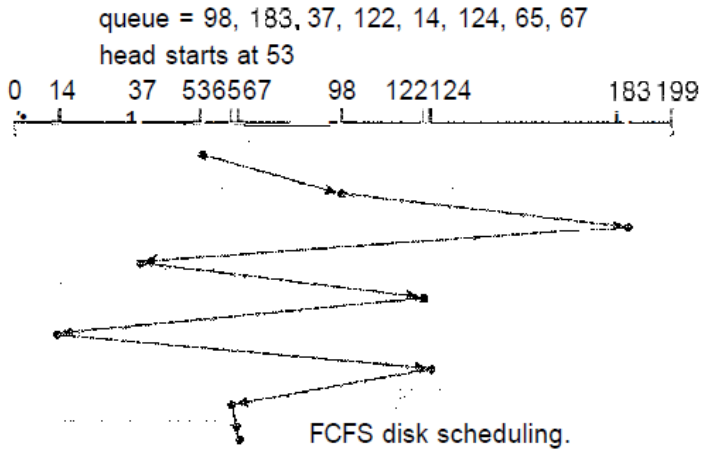
OR

VI. Explain different CPU scheduling algorithms 15

FCFS disc 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 requests for I/O to blocks on cylinders in that order. If the disk head is initially at cylinder 53, it will first move from53 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. The wild swing from 122 to 14 and then back to 124 illustrates the problem with this schedule. If the requests for cylinders 37 and 14 could be serviced together, before or after the requests at 122 and 124, the total head movement could be decreased substantially, and performance could be thereby improved.

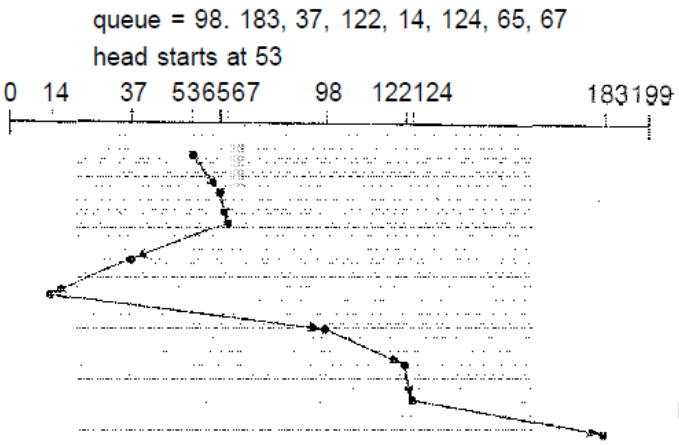
98, 183, 37,122, 14, 124, 65, 67,



FCFS disk scheduling.

SSTF disc Scheduling

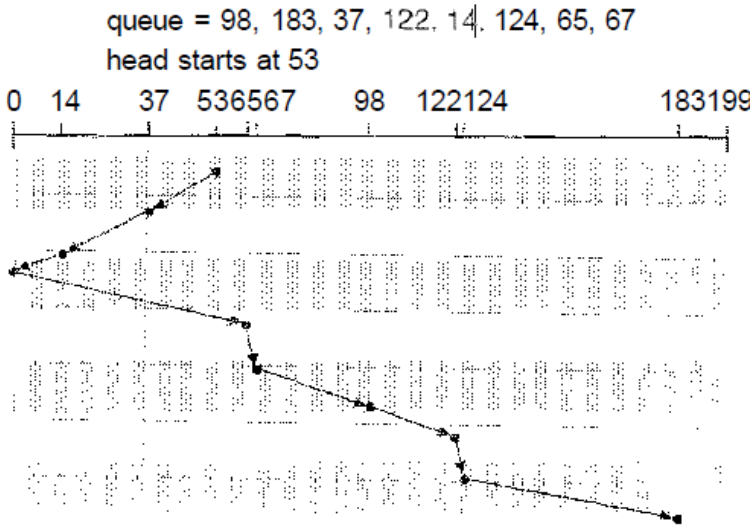
It seems reasonable to service all the requests close to the current head position before moving the head far away to service other requests. This assumption is the basis for the **shortest-seek-time-first (SSTF) algorithm.** The SSTF algorithm selects the request with the minimum seek time from the current head position. Since seek time increases with the number of cylinders traversed by the head, SSTF chooses the pending request closest to the 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 throne at 98, so 37 is served next. Continuing, we service the request at cylinder 14, then 98,122, 124, and finally 183. 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.



SSTF disk scheduling.

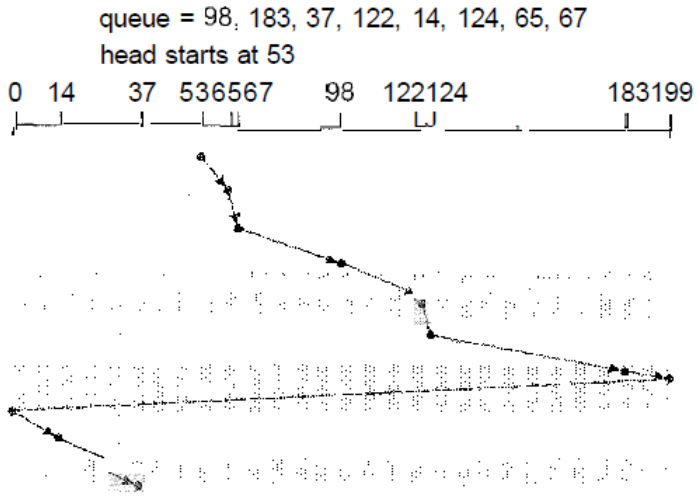
SCAN Scheduling

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 movements reversed, and servicing continues. The head continuously scans back and forth across the disk. The SCAN algorithm is sometimes called the elevator algorithm, since the disk arm behaves just like an elevator in a building, first servicing all the requests going up and then reversing to service requests the other way.



CSCAN Scheduling

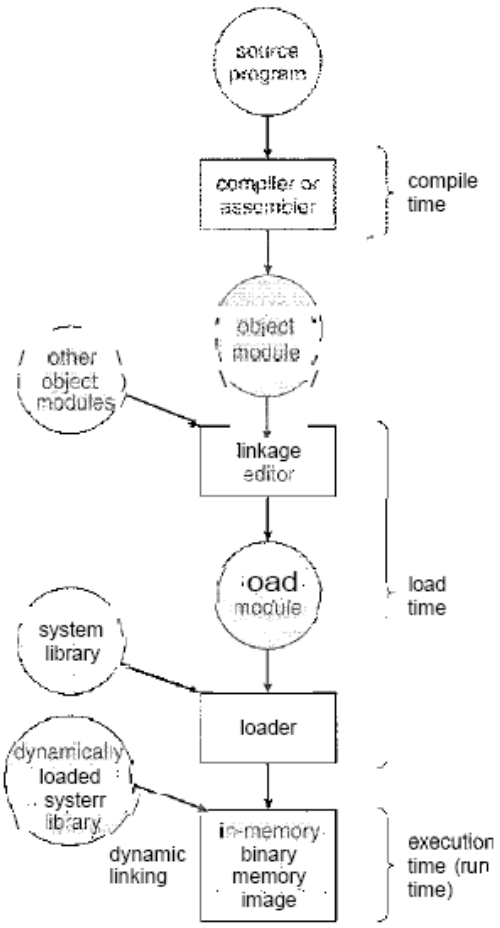
Circular SCAN (C-SCAN) scheduling is a variant of SCAN designed to provide more uniform wait time. Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, however, it immediately returns to the beginning of the disk,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



UNIT – III

VII.

a) Describe the different address binding 9



Compile time: If you know at compile time where the process will reside in memory, then absolute code can be generated. For example, if you know that a user process will reside starting at location R, then the generated compiler code will start at that location and extend up from there. If, at some later time, the starting location changes, then it will be necessary to recompile this code. The MS-DOS .COM-fo.nn.at programs are bound at compile time.

Load time: If it is not known at compile time where the process will reside in memory, then the compiler must generate re-locatable code. In this case, final binding is delayed until load time. If the starting addresses changes, wended only reload the user code to incorporate this changed value.

Execution time: If the process can be moved during its execution from one memory segment to another, then binding must be delayed until runtime. Special hardware must

be available for this scheme to work, as will be discussed in Section 8.1.3. Most general-purpose operating systems use this method.

b) Explain the concept of trashing 6

Thrashing It is a state in which our CPU performs 'productive' work less and 'swapping' more. CPU is busy in swapping pages so much that it cannot respond to user program as much as required. *Why it occurs* In our system Thrashing occurs when there are too much pages in our memory, and each page refers t another page. The real memory shortens in capacity to have all the pages in it, so it uses 'virtual memory'. When each page in execution demands that page that is not currently in real memory (RAM), it places some pages on virtual memory and adjusts the required page on RAM. If CPU is so much busy in doing this task, thrashing occurs.

Thrashing occurs when a computer's virtual memory subsystem is in a constant state of paging, rapidly exchanging data in memory for data on disk, to the exclusion of most application-level processing. This causes the performance of the computer to degrade or collapse. The situation may continue indefinitely until the underlying cause is addressed

- Suppose the pages being actively used by the current threads don't all fit in physical memory.
- Each page fault causes one of the active pages to be moved to disk, so another page fault will occur soon.
- The system will spend all his time reading and writing pages, and won't get much work done.
- This situation is called *thrashing*; it was a serious problem in early demand paging systems.

OR

VIII.

a) Write the following

i. Compare physical address and logical address

ii. Concept of virtual memory 8

i.

- The concept of a logical address space that is bound to a separate physical address space is central to proper memory management.
    - Logical address – generated by the CPU; also referred to as virtual address.
    - Physical address – address seen by the memory unit.
- Logical and physical addresses are the same in compile time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme.

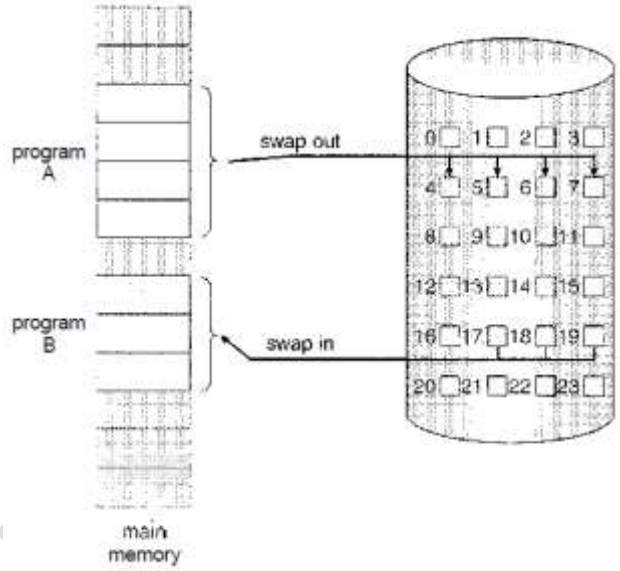
ii.

In a time-sharing system, the operating system must ensure reasonable response time, which is sometimes accomplished through swapping, where processes are swapped in and out of main memory to the disk. A more common method for achieving this goal is virtual memory, a technique that allows the execution of a process that is not completely in memory. The main advantage of the virtual-memory scheme is that it enables users to run programs that are larger than actual 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.

b) Explain demand paging 7

Consider a program that starts with a list of available options from which the user is to select. Loading the entire program into memory results in loading the executable code for all options, regardless of whether an option is ultimately selected by the user or not. An alternative strategy is to initially load pages only as they are needed. This technique is known as demand paging and is commonly used in virtual memory systems. With demand-paged virtual memory, pages are only loaded when they are demanded during program execution; pages that are never accessed are thus never loaded into physical memory.

A demand-paging system is similar to a paging system with swapping where processes reside in secondary memory (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. Since we are now viewing a process as a sequence of pages, rather than as one large contiguous address space, use of the term swapper is technically incorrect. A swapper manipulates entire processes, whereas a pager is concerned with the individual pages of a process. We thus use pager, rather than swapper, in connection with demand paging.



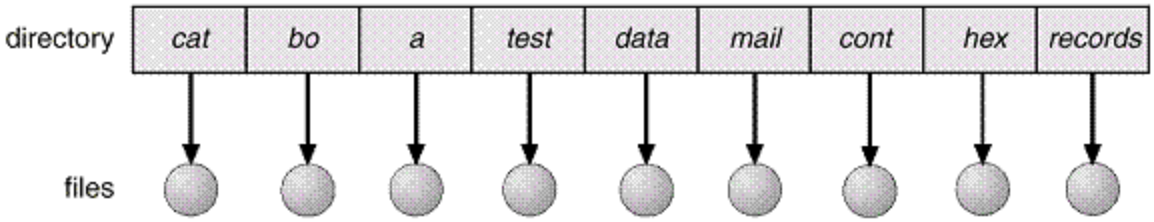
Transfer of a paged memory to contiguous disk space.

UNIT – IV

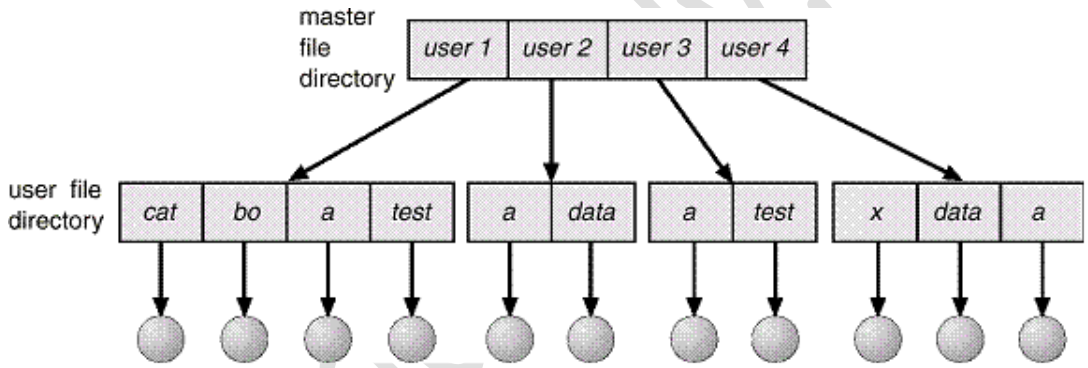
IX.

a) Describe three directory structures 12

Single Level Directory: in this there is a single directory which contains all the Files into. But this is not the best way because all the Files are Stored into the Single Folder So this is very difficult for a user to search a File. There is Only one Directory Which contains the other files So that Many times this is not used because this will Contains Huge Amount of Files and this will Makes difficult for the users to Locate or Find a File



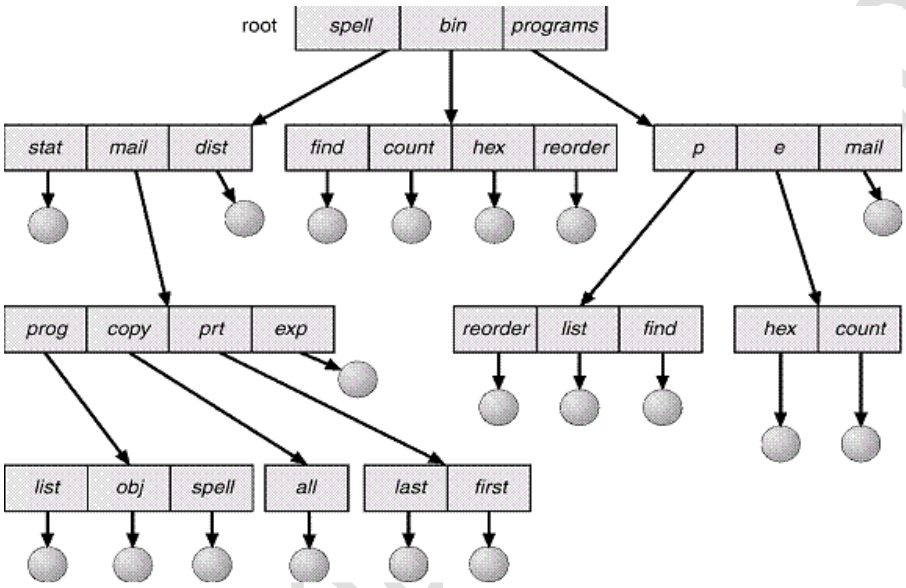
Two Level Directories: in this a Directory also contains a Another Directory and all the Files are Organized into the Sub Directory. In the Two Level a directory also Contains Sub Directory and the Files. Or we can say that a Single Directory will be the Container of Many other files and the Directories So that When a user wants to Access any Directory then he has To Travel all the other Directories and Files from that Directory



Tree structured Directory: Once we have seen how to view a two-level directory as a two-level tree, the natural generalization is to extend the directory structure to a tree of arbitrary height. This generalization allows users to create their own subdirectories and to organize their files accordingly. A tree is the most common directory structure. The tree has a root directory, and every file in the system has a unique path name. A directory 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),
  - as a subdirectory (1).
- Path names can be of two types: **absolute** and **relative**
  - 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.

- With a tree-structured directory system, users can be allowed to access, in addition to their files, the files of other users.
    - For example, user B can access a file of user A by specifying its path names.
    - User B can specify either an absolute or a relative path name.
    - Alternatively, user B can change her current directory to be user A 's directory and access the file by its file names.



b) Write the concept of disc structure 3

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 have 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.

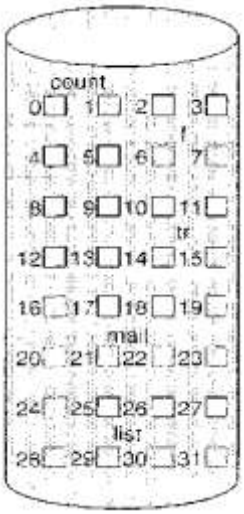
OR

X.

a) Explain different file allocation methods 9

Contiguous Allocation

Contiguous allocation requires that each file occupy a set of contiguous blocks on the disk. Disk addresses define a linear ordering on the disk. With this ordering, assuming that only one job is accessing the disk, accessing block *b* +1 after block *b* normally requires no head movement. When head movement is needed (from the last sector of one cylinder to the first sector of the next cylinder), the head need only move from one track to the next. Thus, the number of disk seeks required for accessing contiguously allocated files is minimal, as is seek time when a seek is finally needed.

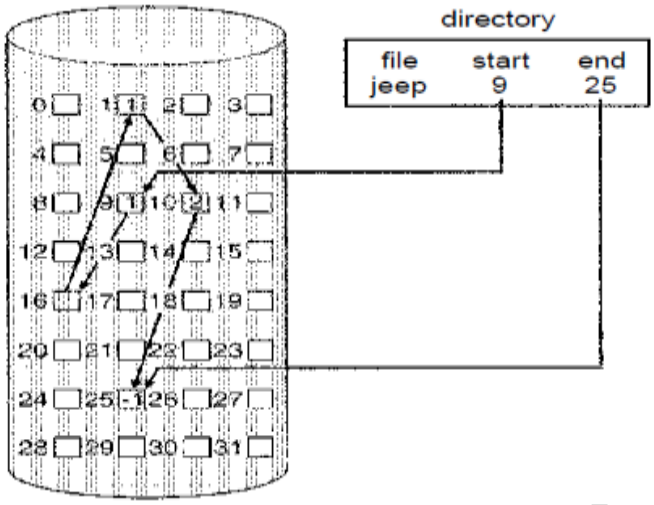


directory

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

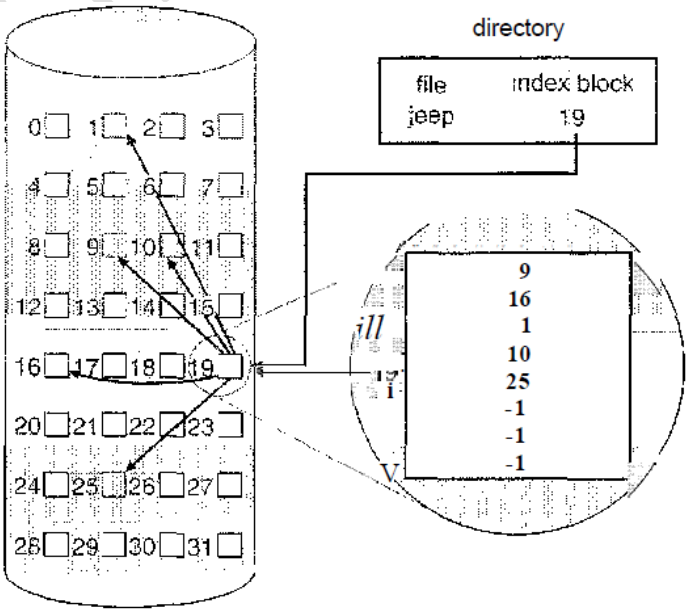
Linked Allocation

Linked allocation solves all problems of contiguous allocation. With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file. For example, a file of five blocks might start at block9 and continue at block 16, then block 1, then block 10, and finally block 25. Each block contains a pointer to the next block. These pointers are not made available to the user. Thus, if each block is 512 bytes in size, and a disk address (the pointer) requires 4 bytes, then the user sees blocks of 508bytes.



Indexed Allocation

Linked allocation solves the external-fragmentation and size-declaration problems of contiguous allocation. However, in the absence of a FAT, linked allocation cannot support efficient direct access, since the pointers to the blocks are scattered with the blocks themselves all over the disk and must be retrieved in order. Indexed allocation solves this problem by bringing all the pointers together into one location: the index block**.** Each file has its own index block, which is an array of disk-block addresses. The i^{th} entry in the index block points to the i^{th} block of the file. The directory contains the address of the index block. To find and read the i^{th} block, we use the pointer in the i^{th} index-block entry.



b) Explain disk management 6

The operating system is responsible for several other aspects of disk management, too. Here we discuss disk initialization, booting from disk, and bad-block recovery.

Disk Formatting

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.

Boot Block

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 bottom 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 "the boot blocks" at a fixed location on the 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; in this case, 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