TED (10)-3071 Reg. No. …….

(REVISION-2010) Signature ……..……

FORTH SEMESTER DIPLOMA EXAMINATION IN ENGINEERING/ TECHNOLIGY- MARCH, 2012

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. Name any four operating systems

   - Windows Operating system
   - Mac OS
   - Unix OS
   - MS DOS

2. Define context switching

   Switching the CPU to another process requires performing a state saveof the current process and a state restore of a different process. This task isknown as a **context switch**

3. Define PCB

   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.

4. Write concept of virtual memory

   Virtual memory is a memory management technique that is implemented using both hardware and software. It mapsmemory addresses used by a program, called virtual addresses, into physical addresses in computer memory. Main

storage as seen by a process or task appears as a contiguous address space or collection of contiguous segments.

5. List any two file organization methods

    - Sequential file organization
    - Indexed file organization

PART – B

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

1. Compare multiprocessing and multiprogramming system

In a **multiprogramming** system there are one or more programs loaded in main memory which are ready to execute. Only one program at a time is able to get the CPU for executing its instructions (i.e., there is at most one process running on the system) while all the others are waiting their turn.The main idea of multiprogramming is to maximize the use of CPU time. Indeed, suppose the currently running process is performing an I/O task (which, by definition, does not need the CPU to be accomplished). Then, the OS may interrupt that process and give the control to one of the other in-main-memory programs that are ready to execute. In this way, no CPU time is wasted by the system waiting for the I/O task to be completed, and a running process keeps executing until either it voluntarily releases the CPU or when it blocks for an I/O operation.

Multiprocessing sometimes refers to executing multiple processes (programs) at the same time. This might be misleading because we have already introduced the term "multiprogramming" to describe that before. In fact, multiprocessing refers to the *hardware* (i.e., the CPU units) rather than the *software* (i.e., running processes). If the underlying hardware provides more than one processor then that is multiprocessing. Several variations on the basic scheme exist, e.g., multiple cores on one die or multiple dies in one package or multiple packages in one system.Anyway, a system can be both multiprogrammed by having multiple programs running at the same time and multiprocessing by having more than one physical processor.

2. Writes notes on program, process and thread.

   A **program** is a sequence of instructions, written to perform a specified task with a computer. A computer requires programs to function, typically executing the program's instructions in a central processor. The program has an executable form that the computer can use directly to execute the instructions

   Process is a program in execution; Process execution must progress in sequential fashion.

   A process includes:

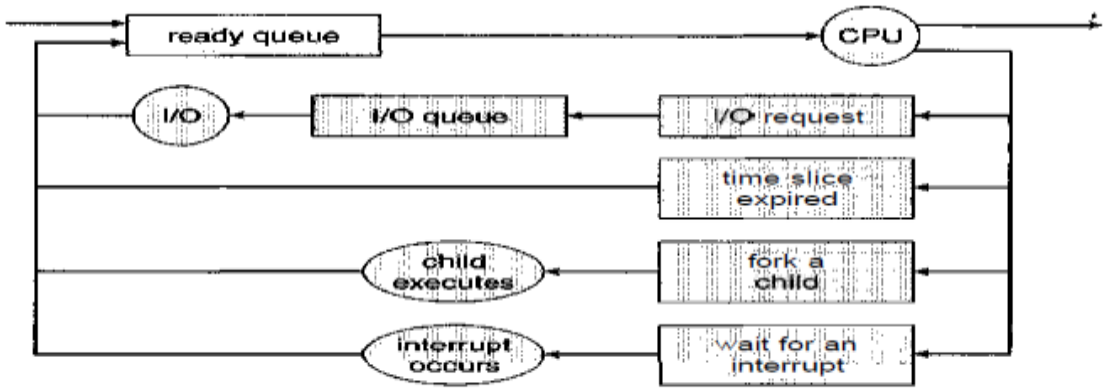
   - program counter - representing the current activity
   - Stack -containing temporary data such as method parameters, return addresses, and local variables
   - Heap - memory that is dynamically allocated during runtime
   - Data section - which contains the global variables

   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. Explain the long term, short term and medium term scheduler with the help of queuing diagram

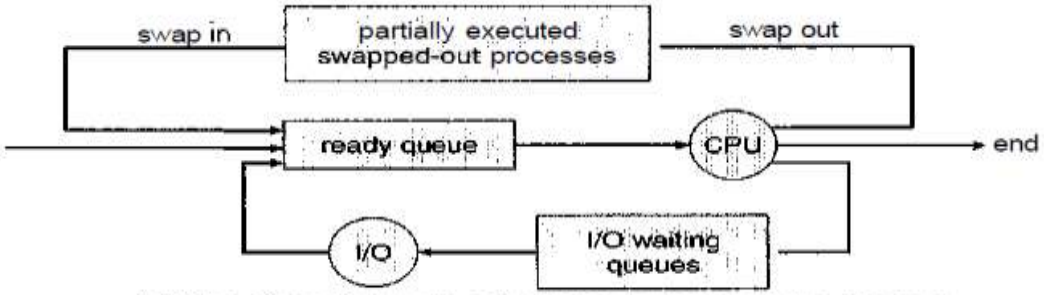
   The **long-term scheduler,** or **job scheduler,** selects processes from this pool and loads them into memory for execution. The **short-term scheduler, or** CPU **scheduler,** selects from among the processes that are ready to execute and allocates the CPU to one of them.

   The primary distinction between these two schedulers lies in frequency of execution. The short-term scheduler must select a new process for the CPU frequently. A process may execute for only a few milliseconds before waiting for an I/O request. Often, the short-term scheduler executes at least once every 100 milliseconds. Because of the short time between executions, the short-term scheduler must be fast.



Queueing-diagram representation of process scheduling.

Some operating systems, such as time-sharing systems, may introduce an additional, intermediate level of scheduling (**medium-term scheduler**). The key idea behind a medium-term scheduler is that sometimes it can be advantageous to remove processes from memory (and from active contention for the CPU) and thus reduce the degree of multiprogramming.



Addition of medium-term scheduling to the queueing diagram.

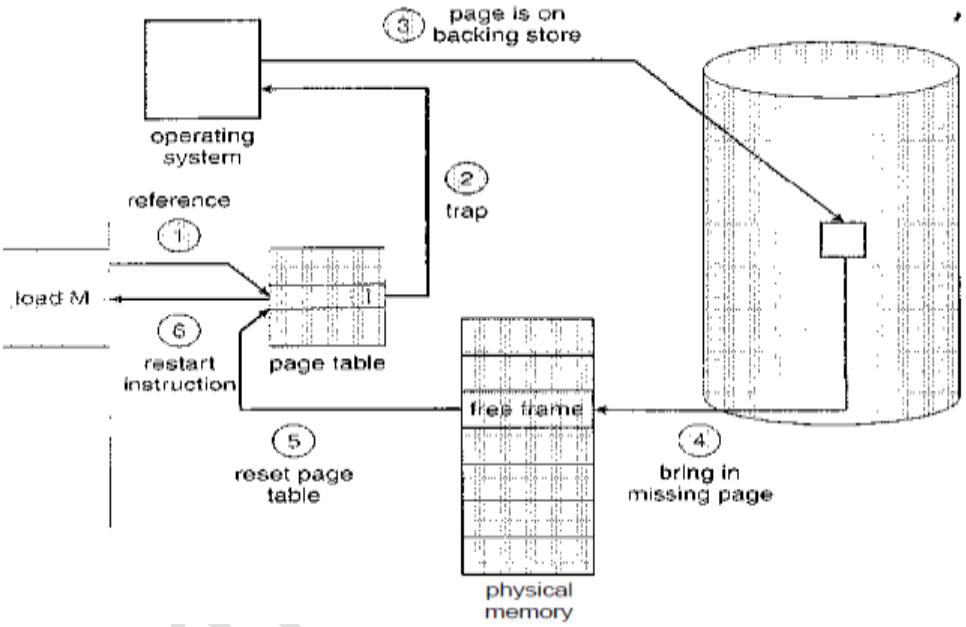
4. Explain the three address binding methods

- **Compile time**: If you know at compile time where the process will residein memory, then absolute code can be generated. For example, if you knowthat a user process will reside starting at location R, then the generatedcompiler code will start at that location and extend up from there. If, atsome later time, the starting location changes, then it will be necessaryto recompile this code. The MS-DOS .COM-fo.nn.at programs are bound atcompile time.
- **Load time:** If it is not known at compile time where the process will residein memory, then the compiler must generate re-locatable code. In this case,final

binding is delayed until load time. If the starting addresses changes, weneed only reload the user code to incorporate this changed value.

- **Execution time:** If the process can be moved during its execution fromone memory segment to another, then binding must be delayed until runtime. Special hardware must be available for this scheme to work, as willbe discussed in Section 8.1.3. Most general-purpose operating systems usethis method.

5. Summarize the action taken by the OS when a page fault occurs



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

6. Write any three services provided by kernel to I/O subsystem

   The kernel's I/O subsystem provides numerous services. Among these is I/O scheduling, buffering, caching, spooling, device reservation, and error handling.

   - **I/O scheduling**:To schedule a set of I/O requests means to determine a good order in which to execute them. The order in which applications issue system calls rarely is the best choice. Scheduling can improve overall system performance, can share device access fairly among processes, and can reduce the average waiting time for I/O to complete.

   - **Buffering**: A **buffer** is a memory area that stores data while they are transferred between two devices or between a device and an application. Buffering is done for three reasons. One reason is to cope with a speed mismatchbetween the producer and consumer of a data stream

   Caching: A cache is a region of fast memory that holds copies of data. Access to the cached copy is more efficient than access to the original. For instance, the instructions of the currently running process are stored on disk, cached in physical memory, and copied again in the CPU's secondary and primary caches

7. Explain any four 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. 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.
   - **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.

PART – C

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

UNIT – I

III. Explain any five operating system components 15

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

Main-Memory Management

Memory is a large array of words or bytes, each with its own address. It is a repository of quickly accessible data shared by the CPU and I/O devices.

- Main memory is a volatile storage device. It loses its contents in the case of system failure.
- The operating system is responsible for the following activities in connections with memory management:
    - Keep track of which parts of memory are currently being used and by whom.
    - Decide which processes to load when memory space becomes available.
    - Allocate and de-allocate memory space as needed.

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.

I/O System Management

- The I/O system consists of:

  – A buffer-caching system

  – A general device-driver interface

  – Drivers for specific hardware devices

Secondary-Storage Management

Since main memory (*primary storage*) is volatile and too small to accommodate all data and programs permanently, the computer system must provide *secondary storage* to back up main memory.

- Most modern computer systems use disks as the principle on-line storage medium, for both programs and data.
- The operating system is responsible for the following activities in connection with disk management:

  – Free space management

  – Storage allocation

  – Disk scheduling

OR

IV.

a) Compare compiler and interpreter 3

The compiler produces an architecture-neutral byte-code output (.class) file that will run on any implementation of the JVM. A compiler may produce assembly code, which is consumed by an assemble

An interpreter is a computer program that directly executes, i.e. performs, instructions written in a programming or scripting language, without previously compiling them into a machine language program

b) Explain functions of a loader 3

- **Loading** – brings the object program into memory for execution

- **Relocation** – modifiesthe 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.

c) Write notes on

i. Batch processing system

ii. Multiprogramming system

iii. Time sharing system 9

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

Multiprogramming system: Multiprogrammingincreases CPU utilization by organizing jobs (code and data) so that the CPU always has one to executeIn 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

Time sharing system: In time-sharing systems, the CPU executes multiple jobs by switching among them, but the switches occur so frequently that the users can interact with each program while it is running. Time sharing requires an interactive (or hands-on) computer system, which provides direct communication between the user and the system

UNIT – II

V.

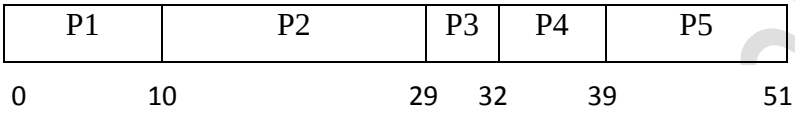
a) Draw the Gantt chart and find the average waiting time in ms of the following process in

1. FCFS 2. SJF 3. Round Robin of time slice 5 ms

PROCESS BURST TIME

P1	10	
P2	19	
P3	3	
P4	7	
P5	12	9

1. FCFS



Waiting time of,

P1 = 0

P2 = 10

P3 = 29

P4 = 32

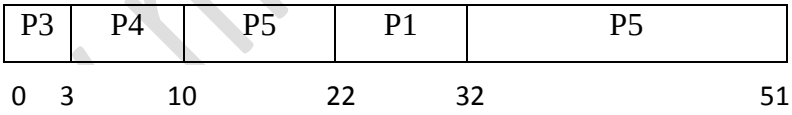
P5 = 39

So,

Average waiting time = (0 + 10 + 29 + 32 + 39) / 5

= 22 ms

2. SJF



Waiting time of,

P1 = 22

P2 = 32

P3 = 0

P4 = 3

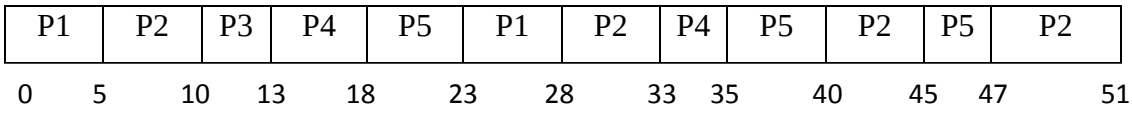
P5 = 10

So,

Average waiting time = (22 + 32 + 0 + 3 + 10) / 5

= 13.4 ms

3. Round Robin



Waiting time of,

P1 = 18

P2 = 32

P3 = 15

P4 = 28

P5 = 35

So,

Average waiting time = (18 + 32 + 15 + 28 + 35) / 5

= 25.6 ms

b) Describe the method of preventing deadlock 6

For a deadlock to occur, each of the four necessary conditions must hold simultaneously. Deadlock prevention is a set of methods for ensuring that at least one of the four necessary conditions cannot hold.

Mutual Exclusion: Removing the mutual exclusion condition means that no process may have exclusive access to a resource. Mutual-exclusion conditions must hold for non-sharable resources and the sharable resources do not require mutually exclusive access. We cannot prevent deadlocks by denying the mutual-exclusion condition: Some resources are intrinsically non-sharable occur.

Hold and Wait: The "hold and wait" conditions may be removed by ensuring that, whenever a process requests a device, it does not hold any other devices.

- One protocol requires each process to request and be allocated all its resources before it begins execution.
- Another protocol requires the processes to release all their allocated resources before requesting additional resources.
- Disadvantages: Low resource utilizationStarvation is possible.

No pre-emption: The third necessary condition is that there is no preemption of resources that have already been allocated. To ensure that this condition does not hold, two protocols are there.

- If a process is holding some resources and requests another resources that cannot be immediately allocated to it, then all the resources currently being held by the process are preempted (released).
- If a process requests some resources, first check whether they are available.
  - If they are available, allocate them.
  - If they are not available and are allocated to some other processes waiting for additional resources, preempt the requested resources from the waiting process and allocate to the requesting process.
  - If the resources are not available and not held by a waiting process, the requesting process must wait. While waiting, some of the resources may be preempted if some other process requests them.

Circular wait condition: To ensure that the circular wait condition never holdsis to impose a total ordering for all resources types.

Let R={ R1, R2,...,Rn} be the set of resource types. Define a one-to-one function: R→N, where N is a set of natural numbers. I.e.assign a unique integer number to each resource type.

- Each process requests resources only in an increasing order of enumeration.
- Whenever a process requests an instance of resource type Rj, it has released any resources Ri such that $F(Ri) \geq F(Rj)$.

OR

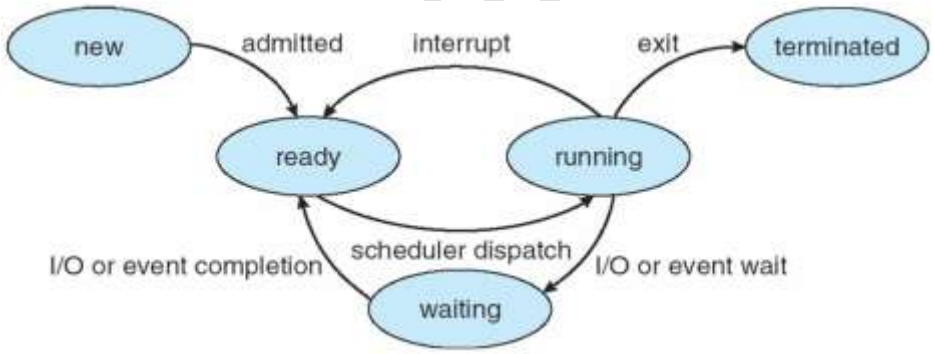
VI.

a) Outline the criteria for evaluating scheduling algorithms 6

- *CPU utilization* – To what percentage CPU is utilized
  - 40% - lightly loaded and 90% - heavily loaded

- *Throughput* – No. of processes that complete their execution per unit time (Degree of multiprogramming)
    - For long processes it could be 1 in an hour and for short processes it could be 10 per sec
- *Turnaround time* –Time interval between the time of submission and completion (Execution time)
    - Includes also waiting times for CPU as well as I/O devices
- *Waiting time* – sum of all the time waiting in *Ready* queue
    - Does not take into account wait time for I/O and I/O operation time
- *Response time* – amount of time it takes from the time of submission of a job until the first response is produced
    - In time shared systems small turn-around time may not be enough
    - Response time should be small. It is the time taken to start responding. Does not include time to output that response

b) With a neat diagram explain the state of a process 6



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

c) Explain critical section problem 3

Consider a system consisting of *n* processes $\{P_1, P_2, ..., P_{n-1}\}$. Each processhas 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.

UNIT – III

VII.

a) Explain first fit, best fit and worst fit allocation strategies 6

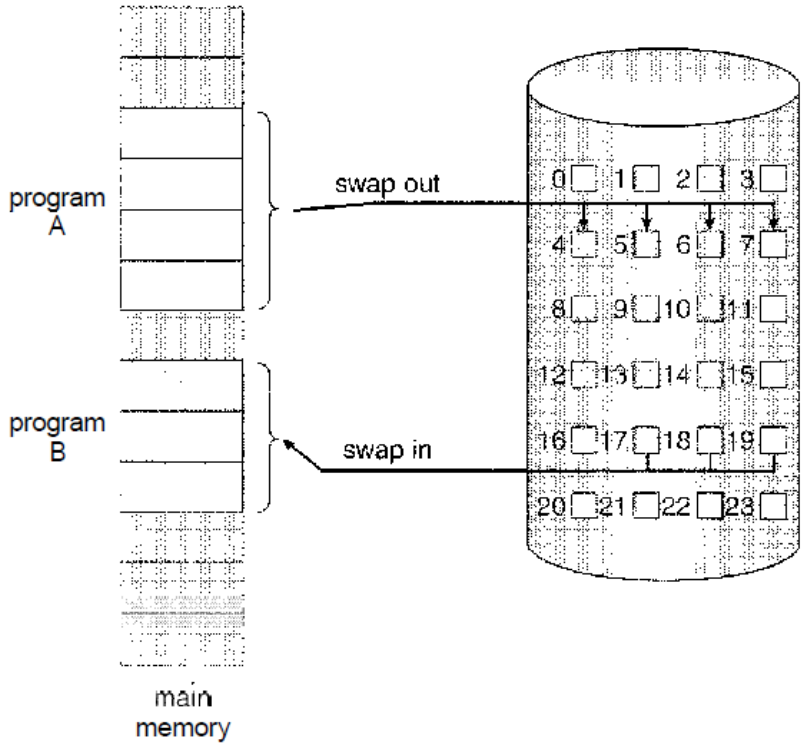
- **First fit:** Allocate the *first* hole that is big enough. Searching can start eitherat the beginning of the set of holes or where the previous first-fit searchended. We can stop searching as soon as we find a free hole that is large enough.
- **Best fit:** Allocate the *smallest* hole that is big enough. We must search theentire list, unless the list is ordered by size. This strategy produces thesmallest leftover hole.
- **Worst fit:** Allocate the *largest* hole. Again, we must search the entire list,unless it is sorted by size. This strategy produces the largest leftover hole,which may be more useful than the smaller leftover hole from a best-fitapproach.

b) Illustrate demand paging 9

Consider a program that startswith a list of available options from which the user is to select. Loading theentire program into memory results in loading the executable code for alloptions, regardless of whether an option is ultimately selected by the user ornot. 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 virtualmemory systems. With

demand-paged virtual memory, pages are only loadedwhen they are demanded during program execution; pages that are neveraccessed are thus never loaded into physical memory.

A demand-paging system is similar to a paging system with swappingwhere processes reside in secondary memory (usually a disk).When we want to execute a process, we swap it into memory. Rather thanswapping the entire process into memory, however, we use a **lazy swapper.** Alazy 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 onelarge contiguous address space, use of the term *swapper* is technically incorrect.A swapper manipulates entire processes, whereas a **pager** is concerned withthe individual pages of a process. We thus use *pager*, rather than *swapper,* inconnection with demand paging.



Transfer of a paged memory to contiguous disk space.

OR

VIII.

a) Describe any three page replacement algorithms 9

In doing so, weuse the reference string

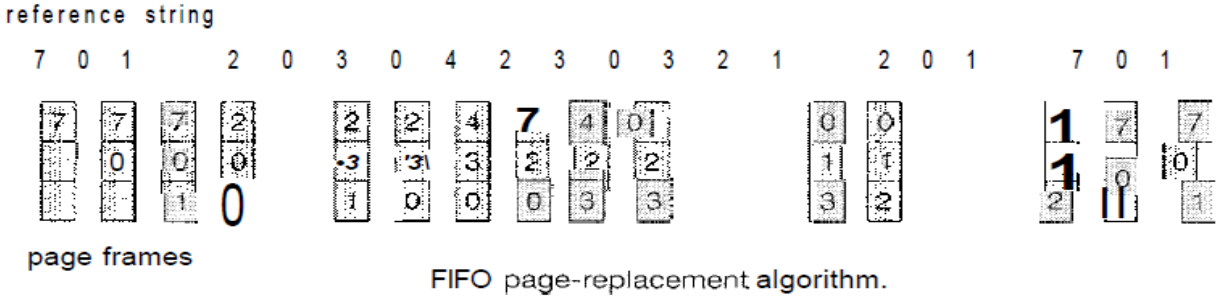
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2,1, 2, 0, 1, 7, 0,1

For a memory with three frames,

FIFO Page Replacement

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 emptyframes. The next reference (2) replaces page 7, because page 7 was brought infirst. Since 0 is the next reference and 0 is already in memory, we have no faultfor this reference. The firstreference to 3 results in replacement of page 0, sinceit is now first in line. Because ofthis replacement, the next reference, to 0, willfault. Page 1 is then replaced by page 0. This process continues as shown inFigure. Every time a fault occurs, we show which pages are in our threeframes. There are 15 faults altogether.

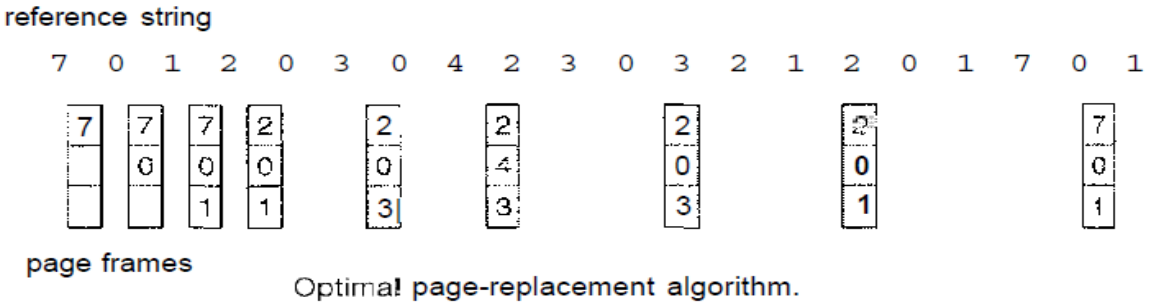
The FIFO page-replacement algorithm is easy to understand and program.However, its performance is not always good. On the one hand, the pagereplaced may be an initialization module that was used a long time ago and isno longer needed. On the other hand, it could contain a heavily used variablethat was initialized early and is in constant use.



FIFO page-replacement algorithm.

Optimal Page Replacement

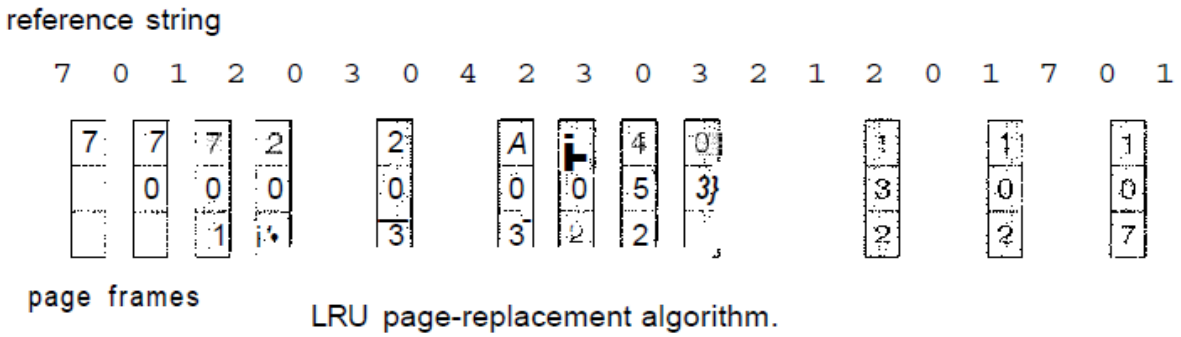
The first threereferences cause faults that fill the three empty frames. The reference to page2 replaces page 7, because 7 will not be used until reference 18, whereas page0 will be used at 5, and page 1 at 14. The reference to page 3 replaces page1, as page 1 will be the last of the three pages in memory to be referencedagain. With only nine page faults, optimal replacement is much better than aFIFO algorithm, which resulted in fifteen faults. In fact, no replacement algorithm can process this referencestring in three frames with fewer than nine faults.



Optimal page-replacement algorithm.

LRU Page Replacement

The result of applying LRU replacement to our example reference string isshown in Figure. The LRU algorithm produces 12 faults. Notice that thefirst 5 faults are the same as those for optimal replacement. When the referenceto page 4 occurs, however, LRU replacement sees that, of the three frames inmemory, page 2 was used least recently. Thus, the LRU algorithm replaces page2, not knowing that page 2 is about to be used. When it then faults for page*2*, the LRU algorithm replaces page 3, since it is now the least recently used ofthe three pages in memory. Despite these problems, LRU replacement with 12faults is much better than FIFO replacement with 15.

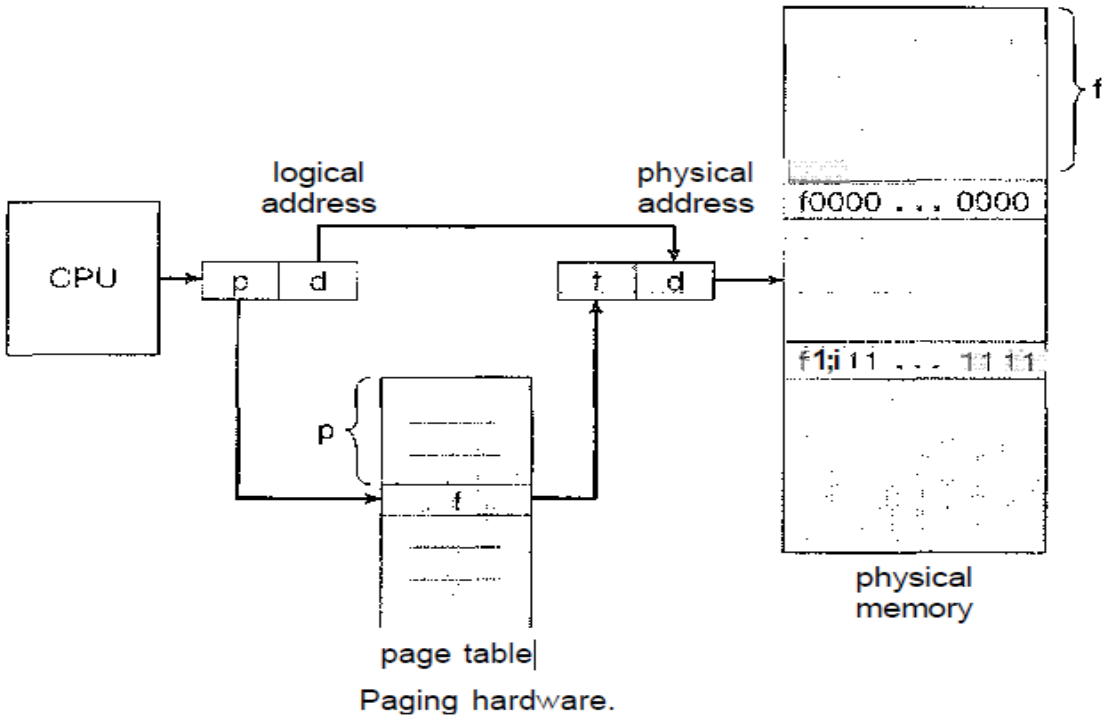


LRU page-replacement algorithm.

b) Explain paging with paging hardware diagram 6

Paging is a memory-management scheme that permits the physical addressspace of a process to be noncontiguous. Paging avoids the considerableproblem of fitting memory chunks of varying sizes onto the backing store; mostmemory-management schemes used before the introduction of paging suffered from this problem. The problem arises because, when some code fragments ordata residing in main memoryneed to be swapped out, space must be foundon the backing store. The backing storealso has the fragmentation problemsdiscussed in connection with main memory; except that access is much slower,so compaction is impossible. Because of its

advantages over earlier methods,paging in its various forms is commonly used in.most operating systems.



Paging hardware.

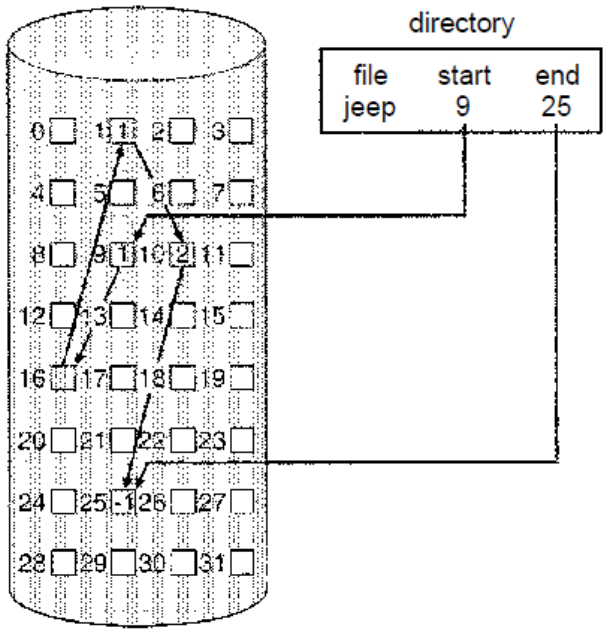
UNIT – IV

IX.

a) Explain linked and indexed file allocation methods 8

Linked file allocation

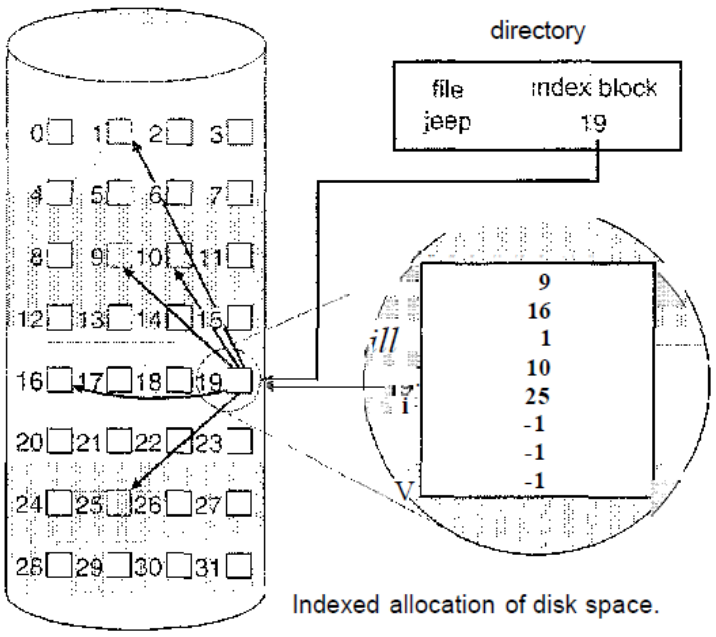
With linkedallocation, each file is a linked list of disk blocks; the disk blocks may bescattered anywhere on the disk. The directory contains a pointer to the firstand last blocks of the file. For example, a file of five blocks might start at block9 and continue at block 16, then block 1, then block 10, and finally block 25. Each block contains a pointer to the next block. These pointersare not made available to the user. Thus, if each block is 512 bytes in size, anda disk address (the pointer) requires 4 bytes, then the user sees blocks of 508bytes



Linked allocation of disk space.

Indexed file allocation

Linked allocation solves the external-fragmentation and size-declaration problemsof contiguous allocation. However, in the absence of a FAT, linkedallocation cannot support efficient direct access, since the pointers to the blocksare scattered with the blocks themselves all over the disk and must be retrievedin order. **Indexed allocation** solves this problem by bringing all the pointerstogether 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 directorycontains 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.



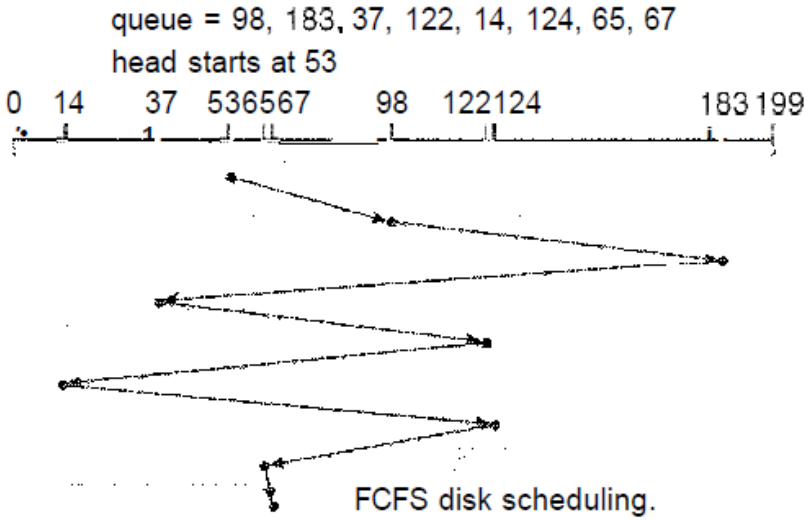
Indexed allocation of disk space.

b) Illustrate FCFS and SSTF disc scheduling algorithms 7

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 notprovide the fastest service. Consider, for example, a disk queue with requestsfor I/O to blocks on cylindersin 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 headmovement of 640 cylinders. This schedule is diagrammed in Figure. The wild swing from 122 to 14 and then back to 124 illustrates the problemwith 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 movementcould be decreased substantially, and performance could be thereby improved.

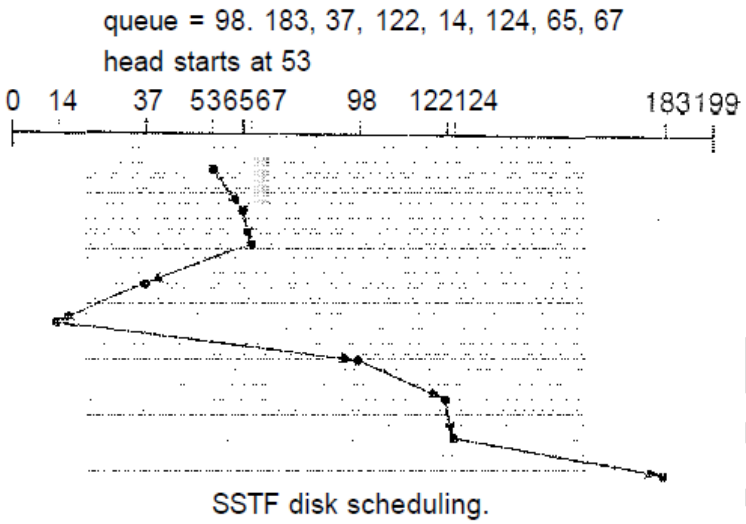
98, 183, 37,122, 14, 124, 65, 67,



FCFS disk scheduling.

SSTF disc Scheduling

It seems reasonable to service all the requests close to the current head positionbefore moving the head far away to service other requests. This assumption isthe basis for the **shortest-seek-time-first (SSTF) algorithm.** The SSTF algorithmselects 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 headposition (53) is at cylinder 65. Once we are at cylinder 65, the next closestrequest is at cylinder 67. From there, the request at cylinder 37 is closer than theone 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 resultsin a total head movement of only 236 cylinders—little more than one-third ofthe distance needed for FCFS scheduling of this request queue. This algorithmgives a substantial improvement in performance.



SSTF disk scheduling.

OR

X.

a) Explain the way in which the OS performs disc management 9

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. 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 modes) and an initial empty directory.

Boot Block

Since ROM is read only, it cannot be infected by a computer virus. The problem isthat changing this bootstrap code requires changing the ROM, hardware chips.For this reason, most systems store a tiny bootstrap loader program in the bootROM

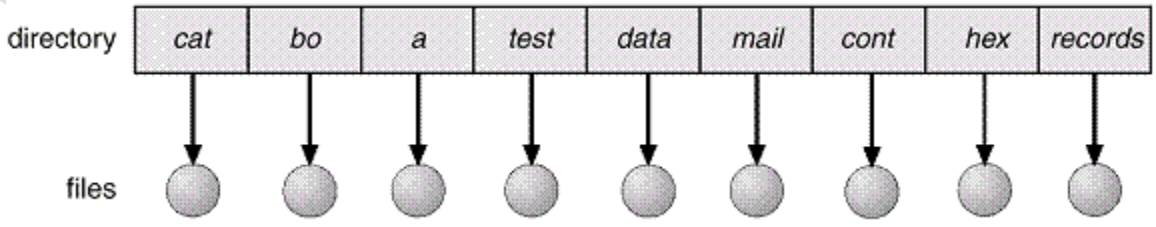
whose only job is to bring in a full bootstrap program from disk. Thefull bootstrap program can be changed easily: A new version is simply writtenonto the disk. The full bootstrap program is stored in "the boot blocks" at a fixed location on the disk. A disk that has a boot partition is called a boot disk or system disk. The code in the boot ROM instructs the disk controller to read the boot blocks into memory (no device drivers are loaded at this point) and then starts executing that code. The full bootstrap program is more sophisticated than the bootstrap loader in the boot ROM; it is able to load the entire operating system from a non-fixed location on disk and to start the operating system running. Even so, the full bootstrap code may be small.

Bad Blocks

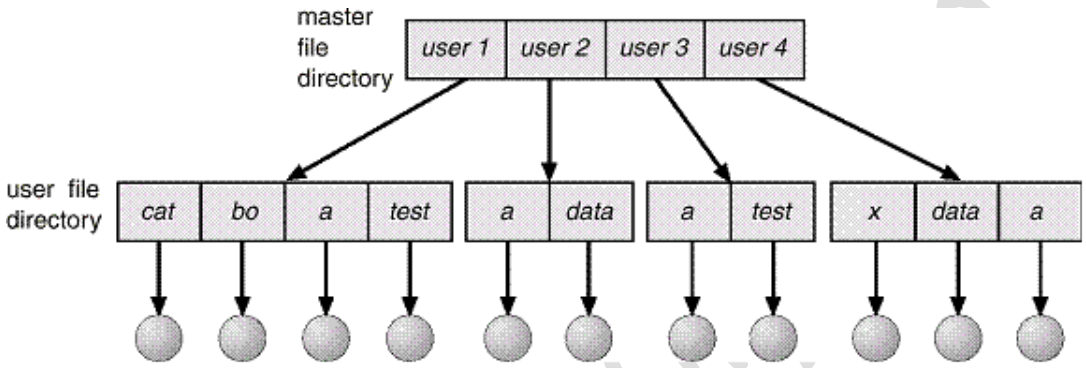
Because disks have moving parts and small tolerances (recall that the diskhead flies just above the disk surface), they are prone to failure. Sometimes thefailure is complete; in this case, the disk needs to be replaced and its contentsrestored from backup media to the new disk. More frequently, one or moresectors become defective. Most disks even come from the factory with badblocks. Depending on the disk and controller in use, these blocks are handledin a variety of ways

b) Explain three directory structures 6

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.