TED (10)-3071 Reg. No. ………………………….

(REVISION-2010) Signature ………………………

# FORTH SEMESTER DIPLOMA EXAMINATION IN ENGINEERING/ TECHNOLIGY- OCTOBER, 2013

**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 function of 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

2. List four advantages of multithreaded programming

- Responsiveness
- Resource sharing.
- Economy.
- Utilization of multiprocessor architectures.

3. Differentiate CPU bound and I/O bound process

- An **I/O-bound process** is one that spends more of its time doing I/O than it spends doing computations.
- A **CPU-bound process,** in contrast, generates I/O requests infrequently, using more of its time doing computations.

4. Give two solutions for external fragmentation

   One solution to the problem of external fragmentation is compaction. Another possible solution to the external-fragmentation problem is to permit the logical address space of the processes to be noncontiguous, thus allowing a process to be allocated physical memory wherever the latter is available.

5. List any four operations on files

   - Creating a file
   - Writing a file
   - Reading a file
   - Repositioning within a file

PART – B

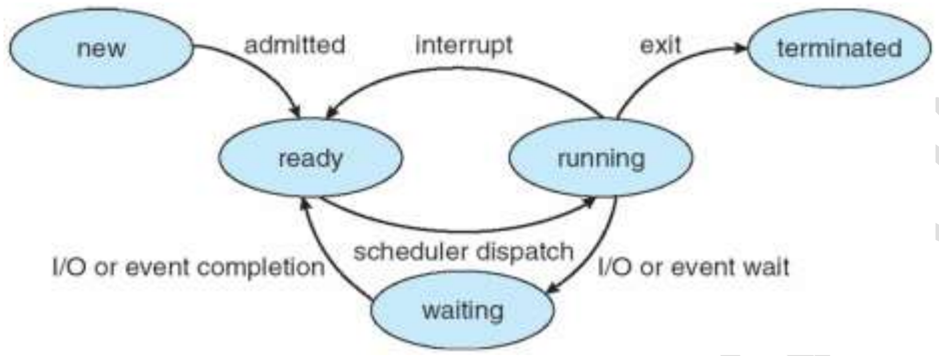
II. Answer *any five* questions. Each question carries 6 marks

1. Compare compiler and interpreter

| Interpreter | Compiler |
|---|---|
| Translates program one statement at a time. | Scans the entire program and translates it as a whole into machine code. |
| It takes less amount of time to analyze the source code but the overall execution time is slower. | It takes large amount of time to analyze the source code but the overall execution time is comparatively faster. |
| No intermediate object code is generated, hence are memory efficient. | Generates intermediate object code which further requires linking, hence requires more memory. |
| Continues translating the program until the first error is met, in which case it stops. Hence debugging is easy. | It generates the error message only after scanning the whole program. Hence debugging is comparatively hard. |

| | |
|---|---|
| Programming language like Python, Ruby uses interpreters. | Programming language like C, C++ use compilers. |

2. With a neat diagram explain the states of a process



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

3. Write the necessary condition for the occurrence of deadlock

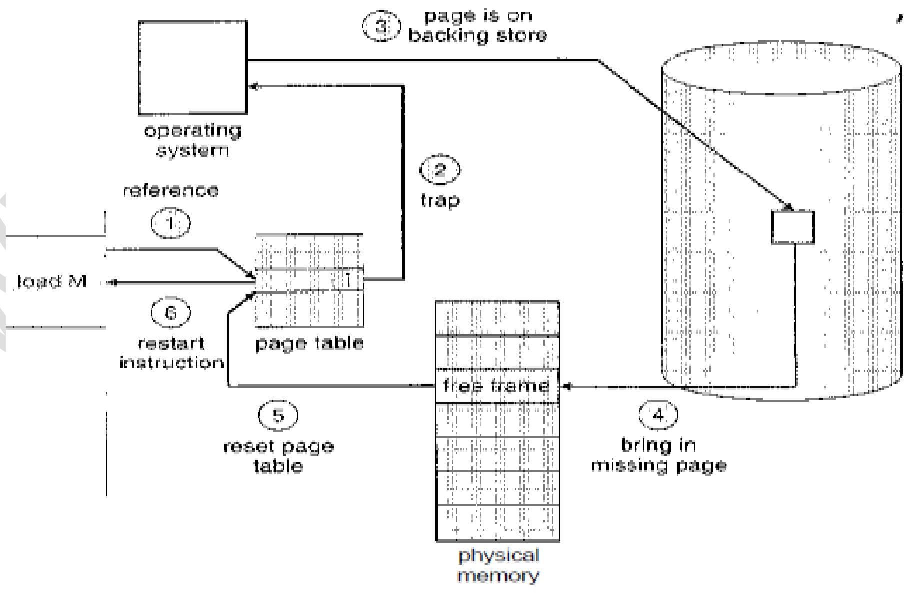
Necessary conditions for deadlock to occur are:

- **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 beforcibly 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 byP1.

4. Explain memory allocation strategies
   - **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 largeenough.
   - **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.

5. Describe 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.

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. Distinguish between SCAN and C-SCAN algorithms

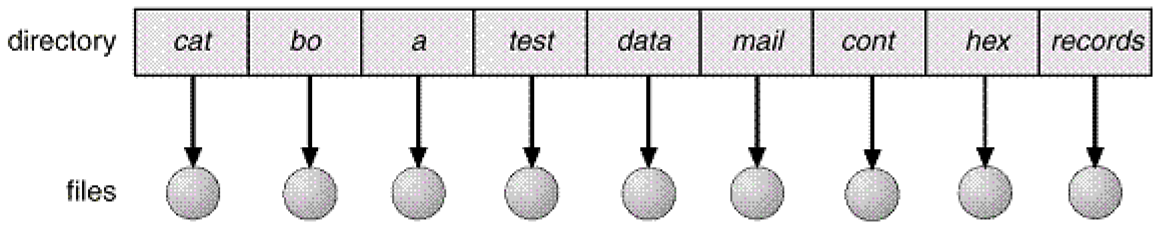
In the SCAN algorithm, the disk arm starts at one end of the disk and movestoward the other end, servicing requests as it reaches each cylinder, until it getsto the other end of the disk. At the other end, the direction of head movementis reversed, and servicing continues. The head continuously scans back andforth across the disk. The SCAN algorithm is sometimes called the elevatoralgorithm, since the disk arm behaves just like an elevator in a building, firstservicing all the requests going up and then reversing to service requests theother way.

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

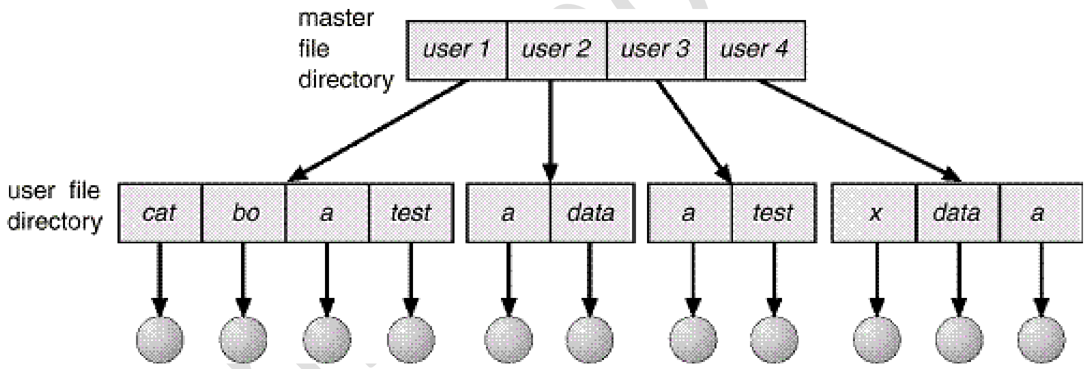
7. Summarize the different directory structures

**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

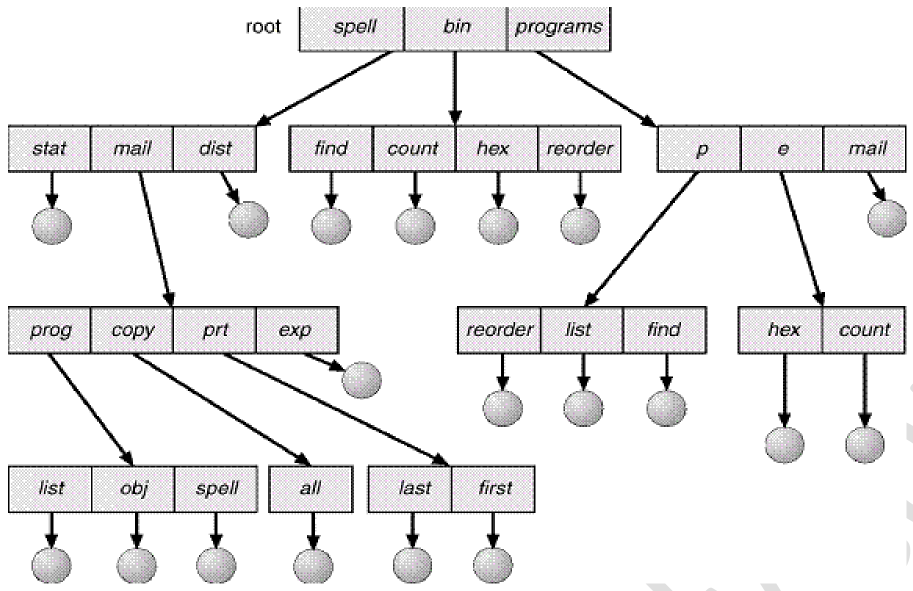
Huge Amount of Files and this will Makes difficult for the users to Locate or Find a File



**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.



PART – C

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

UNIT – I

III.

a) Explain the memory management, process management and file management components of OS 9

**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.

**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

**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

b) Write notes on batch processing and real time system 6

**Batch processing system**

- A batch processing system is one where data are collected together in a batch before processing starts.
- User prepares his program (job) offline and submits it to the computer center.
- The job was submitted in the form of punch cards.
- The computer operator batches the similar jobs and loads this batch of programs into the computer at one time where they are executed one after another.
- 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.

**Real time systems**

- Often used as a control device in a dedicated application such as controlling scientific experiments, medical imaging systems, industrial control systems, and some display systems
- Well-defined fixed-time constraints
- Real-Time systems may be either *hard* or *soft* real-time

OR

IV.

a) Explain I/O management and network management components of OS 6

**I/O System Management**

- The I/O system consists of:
  – A buffer-caching system
  – A general device-driver interface
  – Drivers for specific hardware devices

**Network management**

- A *distributed* system is a collection processors that do not sharememory or a clock

        - Each processor has its own local memory
    - The processors in the system are connected through acommunication 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

b) Compare multiprogramming, time sharing and multiprocessing systems 9

**Multiprogramming system**

- The OS gives each process a certain timeslice (quantum)to run.
- Control is passed to another process if:
    - running process ends before timeslice expires
        - Running process leaves the system.
    - running process needs and I/O operation
        - Running process joins the I/O device queue,
    - Time slice expires
        - Running process goes back to the CPU queue.

**Time-Sharing Systems**

- The CPU is multiplexed among several jobs that are keptin memory and on disk (the CPU is allocated to a job onlyif the job is in memory)
- A job is swapped in and out of memory to the disk
- On-line communication between the user and the system is provided
    - When the operating system finishes the execution of one command, it seeks the next "control statement" from the user's keyboard
- On-line system must be available for users to access data and code

**Multiple-processor systems**

- Symmetric multiprocessing
    - Each processor runs an identical copy of the operating system.
    - Many processes can run at once without performance deterioration.
- Asymmetric multiprocessing

    - Each processor is assigned a specific task;master processor schedules and allocates workto slave processors.
    - More common in extremely large systems

UNIT – II

V.

a) Differentiate preemptive and non-preemptive scheduling 6

- Non-preemptive scheduling: A process runs to completion when scheduled
- Preemptive scheduling: A process may be preempted for another process which may be scheduled. A set of processes are processed in an overlapped manner
- CPU scheduling decisions occur when a process:
  1) Switches from running to waiting state.
  2) Switches from running to ready state (interrupt occurs)
  3) Switches from waiting to ready state (ex. after completion of I/O)
  4) Terminates
- Scheduling under 1, 4 is non-preemptive
    - FCFS – First come first served
    - SJF – shortest job first  and SRTF – shortest remaining time first
    - Priority scheduling
- Scheduling under 2 and 3 is preemptive
    - SJF
    - Priority scheduling
    - Round robin
    - Multilevel queue
    - Multilevel feedback queue

b) Describe deadlock prevention methods 9

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 utilization Starvation 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) ≥ F(Rj).

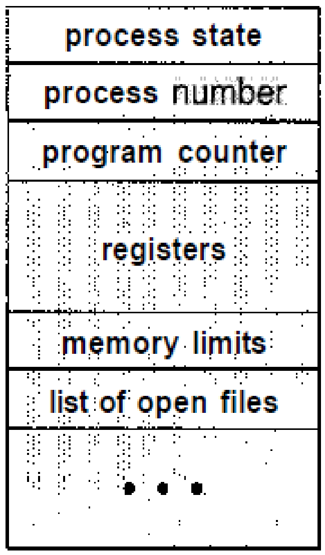
OR

VI.

a) Explain general structure of PCB 5

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 ofinformation 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).

b) Describe multilevel queue and feedback queue scheduling 5

**Multilevel queue**

- This scheduling partitions the ready queue into several separate queues.
- Ex: Ready queue can be logically divided into separate queues based on the idea that jobs can be categorized as:foreground (interactive), background (batch)
- Assign high priority for type 1 jobs – externally
- These two categories have different response time requirements – make 2 queues
- Each queue has its own scheduling algorithm: foreground – RR, background – FCFS
- Method is complex but flexible

**Feedback queue scheduling**

- Preemptive scheduling with dynamic priorities
- A process can move between the various queues
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue

- method used to determine which queue a process will enter when that process needs service
- method used to determine when to upgrade process
- method used to determine when to demote process

c) Illustrate resource allocation graph 5

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, Pi,,.., P,,\, the set consisting of all the active processes in the system, and R = {R[, R2, •••/ Rm}, the set consisting of all resource types in the system.

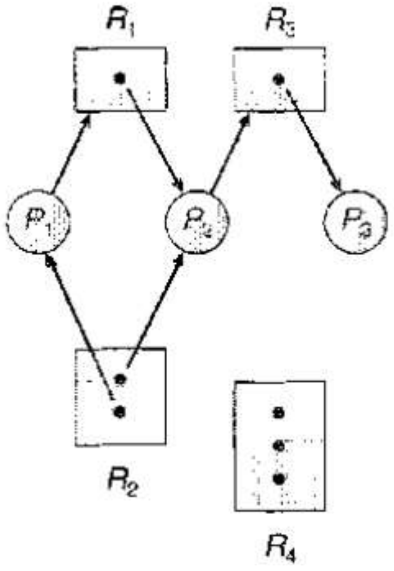
The resource-allocation graph shown in Figure depicts the following situation.

The sets P, *R*, and £:

- *P={PhP2/P?,}*
- *R*= {/?!, *RZ,R3*, R;}
- £ = {p, *_>Ru P2 _>* R3/ R, *_>p2f R2 _>* P2/ *R2 _>* p.,, *R3* ->P3 }

Resource instances:

- One instance of resource type R1
- Two instances of resource type R2
- One instance of resource type *R3*
- Three instances of resource type *R4*



Resource allocation graph

Process states:

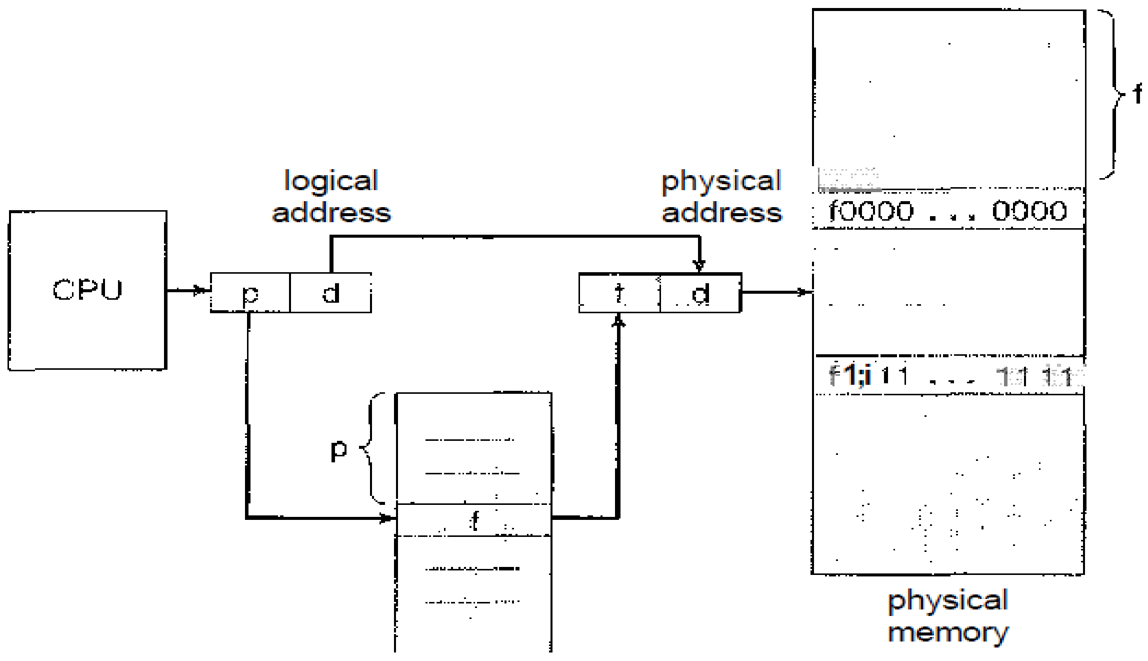
- Process P1is holding an instance of resource type R2 and is waiting for an instance of resource type R1.
- Process Pn is holding an instance of R1 and an instance of R2 and is waiting for an instance of R3.
- Process P3 is holding an instance of R3.

UNIT – III

VII.

a) Explain paging scheme 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.

b) Illustrate any 3 page replacement algorithms 9

In doing so, weuse the reference string

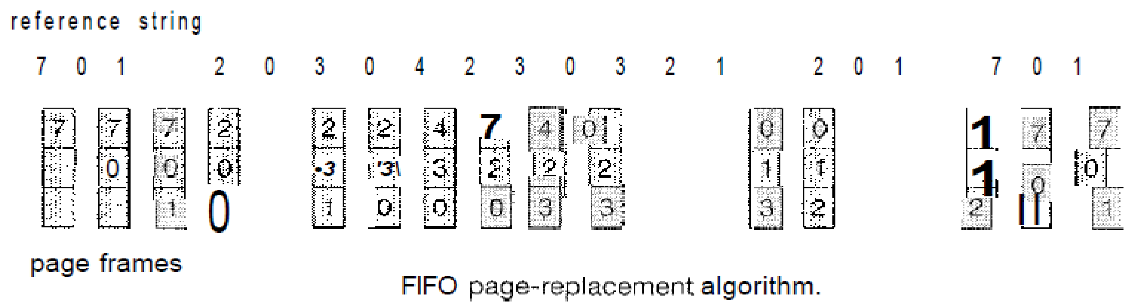
7, 0,1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2,1, 2, 0, 1, 7, 0,1

For a memory with three frames,

**FIFO Page Replacement**

For our example reference string, our three frames are initially empty. Thefirst 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.

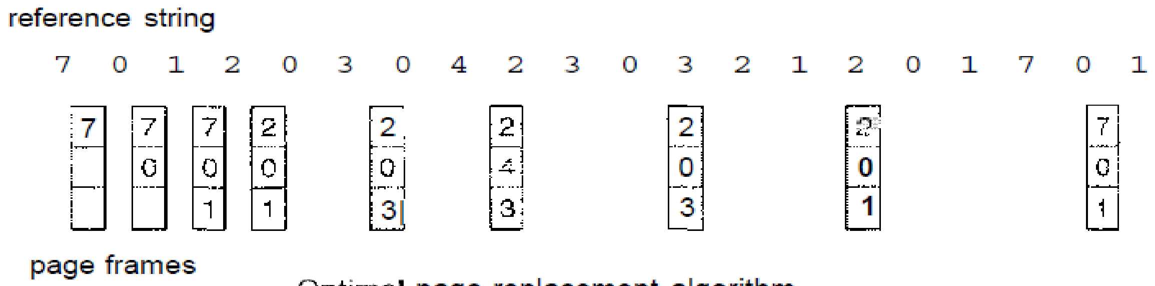
The FIFO page-replacement algorithm is easy to understand and program.However, its performance is not always good. On the one hand, the pagereplaced may be an initialization module that was used a long time ago and isno longer needed. On the other hand, it could contain a heavily used variablethat was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



FIFO page-replacement algorithm.

## Optimal Page Replacement

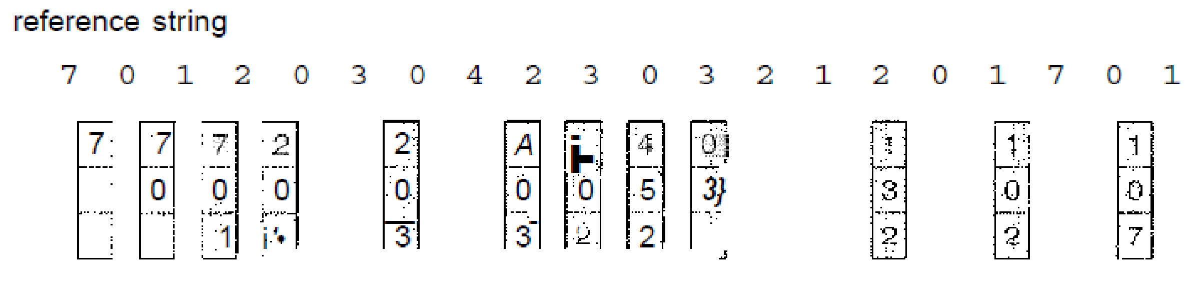
The first threereferences cause faults that fill the three empty frames. The reference to page2 replaces page 7, because 7 will not be used until reference 18, whereas page0 will be used at 5, and page 1 at 14. The reference to page 3 replaces page1, as page 1 will be the last of the three pages in memory to be referencedagain. With only nine page faults, optimal replacement is much better than aFIFO algorithm, which resulted in fifteen faults. (If we ignore the first three,which all algorithms must suffer, then optimal replacement is twiceas good asFIFO replacement.) 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.

OR

VIII.

a) List and compare the different address binding scheme 9

- Compile time. The compiler translates symbolic addresses to re-locatable addresses. If it is not known at compile time where the process will reside in memory, then the compiler must generate re-locatable code. If it is known at compile time where the process will reside in memory, then absolute code can be generated.
- Load time. The loader translates the re-locatable address generated by the compiler to absolute addresses.
- Execution time. Binding delayed until run time if the process can be moved during its execution from one memory segment to another. Execution time binding requires hardware support for address maps (e.g., base and limit registers). Most general-purpose OSs uses this method (Dynamic).
- The compile-time and load-time address-binding methods generate identicallogical and physical addresses. However, the execution-time addresses bindingscheme results in differing logical and physical addresses. In this case,we usually refer to the logical address as a virtual address. We use logical addressand virtual address interchangeably in this text. The set of all logical addressesgenerated by a program is a logical address space; the set of all physicaladdresses corresponding to these

logical addresses is a physical address space.Thus, in the execution-time address-binding scheme, the logical and physicaladdress spaces differ.

b) Explain trashing 3

**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

c) List advantages of segmentation over paging 3

- No internal fragmentation (but: external fragmentation)
- May save memory if segments are very small and should not be combined into one page (e.g. for reasons of protection)
- Segment tables: only one entry per actual segment as opposed to one per page in VM
- Average segment size >> average page size
  $\Rightarrow$ less overhead (smaller tables)

UNIT – IV

IX.

a) Describe any 3 services provided by kernelI/O subsystem 9

- **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.Here is a simple example to illustrate the opportunity. Suppose that a disk arm is near the beginning of a disk and that three applications issue blocking read calls to that disk. Application 1 requests a block near the end of the disk, application 2 requests one near the beginning, and application 3 requests one in the middle of the disk. The operating system can reduce the distance that the disk arm travels by serving the applications in the order 2, 3,1. Rearranging the order of service in this way is the essence of I/O scheduling.

- **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 mismatch between the producer and consumer of a data stream Suppose, for example, that a file is being received via modem for storage on the hard disk. The modem is about a thousand times slower than the hard disk. So a buffer is created in main memory to accumulate the bytes received from the modem. When an entire buffer of data has arrived, the buffer can be written to disk in a single operation. Since the disk write is not instantaneous and the modem still needs a place to store additional incoming data, two buffers are used.

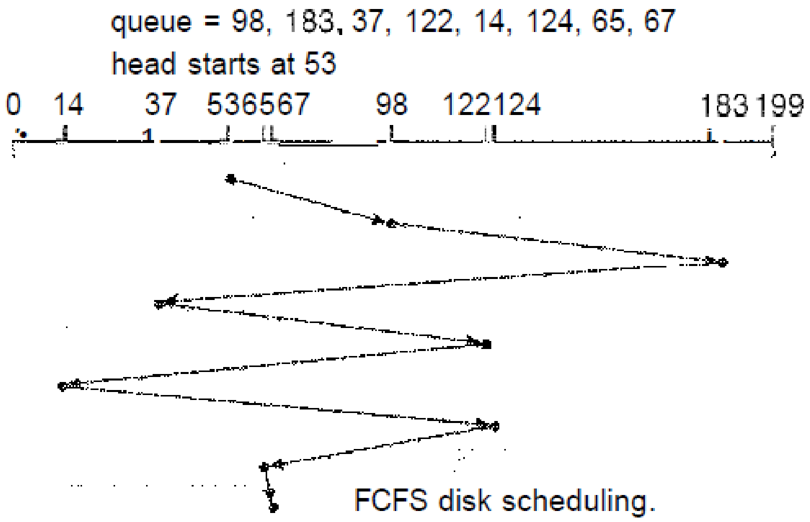
- **Caching:** A cache is a region of fast memory that holds copies of data. Access to the cached copy is more efficient than access to the original. For instance, the instructions of the currently running process are stored on disk, cached in physical memory, and copied again in the CPU's secondary and primary caches. The difference between a buffer and a cache is that a buffer may hold the only existing copy of a data item, whereas a cache, by definition, just holds a copy on faster storage of an item that resides elsewhere.

b) Illustrate FCFS & SSTF disc scheduling algorithms 6

**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.

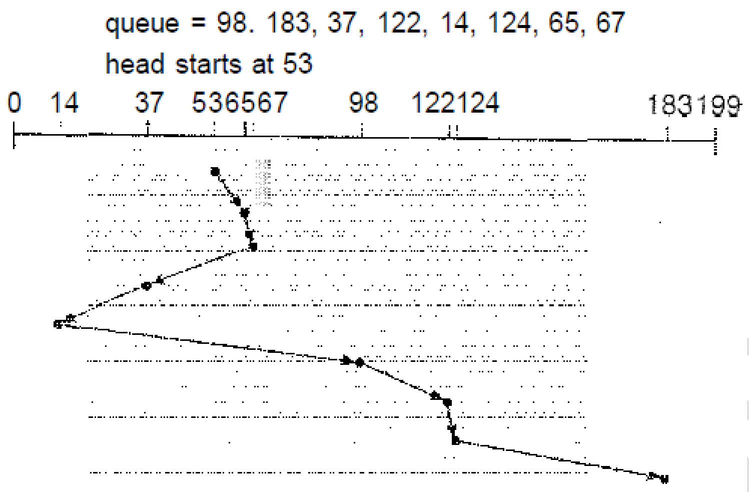
98, 183, 37,122, 14, 124, 65, 67,



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.

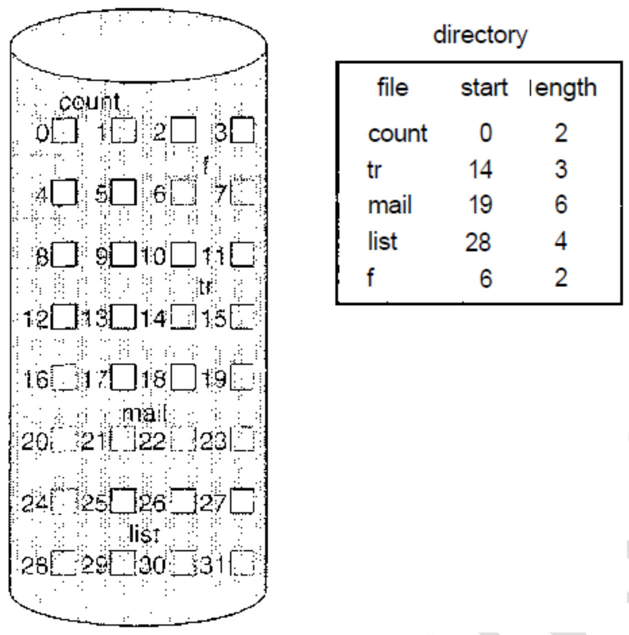
OR

X. Explain different file allocation methods 15

**Contiguous Allocation**

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

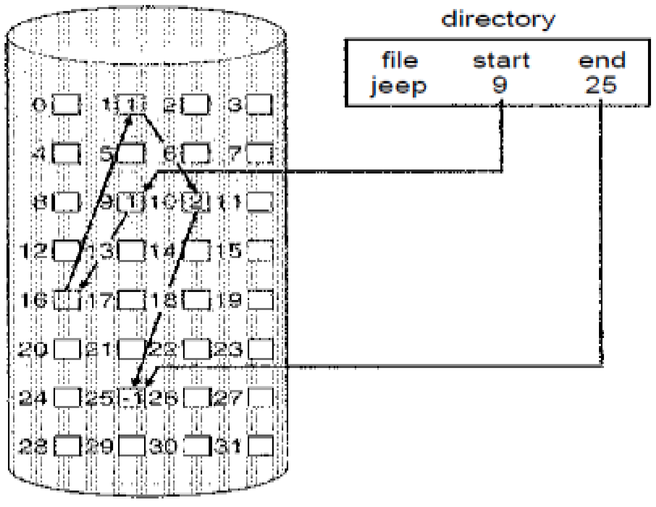
Contiguous allocation of a file is defined by the disk address and length (inblock units) of the first block. If the file is *n* blocks long and starts at location*b,* then it occupies blocks *b, b* + 1, *b* + *2, ...,b* + *n* — 1. The directory entry foreach file indicates the address of the starting block and the length of the areaallocated for this file



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

## Linked Allocation

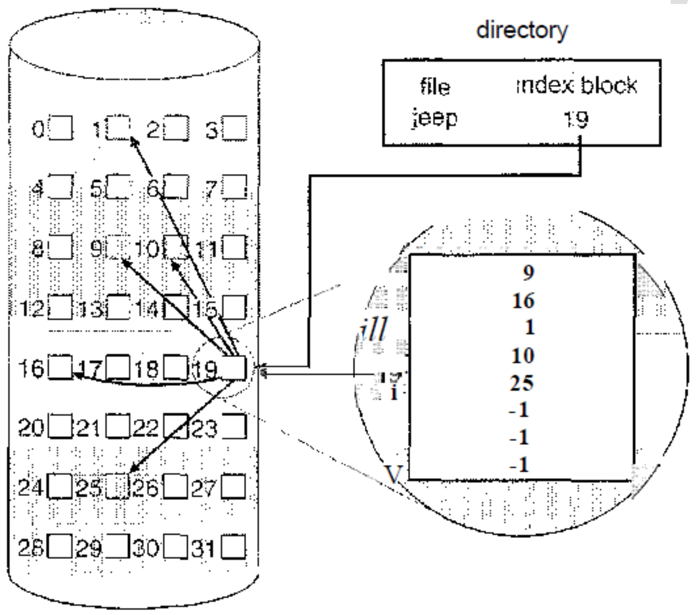
Linked allocation solves all problems of contiguous allocation. With linkedallocation, each file is a linked list of disk blocks; the disk blocks may bescattered anywhere on the disk. The directory contains a pointer to the firstand last blocks of the file. For example, a file of five blocks might start 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.



| file | start | end |
|---|---|---|
| jeep | 9 | 25 |

## Indexed 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 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 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.


directory
file
index block
jeep
19
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
16 17 18 19
20 21 22 23
24 25 26 27
28 29 30 31
9
16
1
10
25
-1
-1
-1