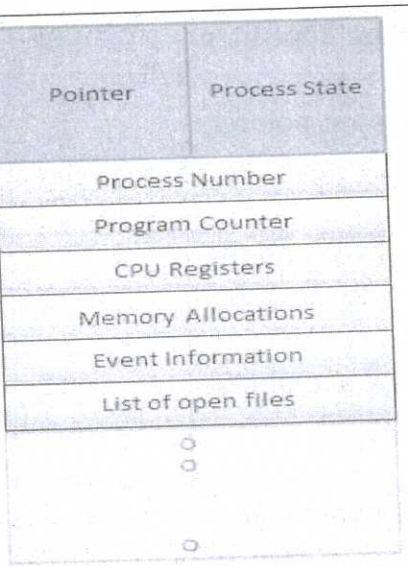
SCORING INDICATOR – (15)4134 OPERATING SYSTEMS			
I	PART A		10
1	System software is a type of computer program that is designed to run a computer's hardware and application programs.	2	
2	A thread is a single sequence stream within a process. Because threads have some of the properties of processes, they are sometimes called lightweight processes. the basic unit of CPU utilization is a thread. A thread has or consists of a program counter (PC), a register set, and a stack space	Any 2 x 1	
3	Process Management, Memory Management, File-System Management, Mass-storage Management, Protection and Security, User-interface etc etc	Any 4 x ½	
4	A process is thrashing if it is spending more time on paging than executing.	2	
5	A thin client is a lightweight computer that is built for remoting into a server. It depends heavily on another computer (its server) to fulfil its computational roles.	2	
II 1	Both are language translators that translate high level language to machine language. Both are machine independent. A compiler is a computer program (or a set of programs) that transforms source code written in a programming language (the source language) into another computer language(the target language), with the latter often having a binary form known as object code Features Compiler Takes entire program as input Intermediate object code is generated Memory requirement : More(Since object code is generated) Program need not be compiled every time Errors are displayed after entire program is checked Faster execution	3+3	6

	Eg: C compiler Interpreter Interpreter takes single instruction as input No intermediate object code is generated Memory requirement is less Every time higher level program is converted into lower level program Errors are displayed for every instruction interpreted(if any) Line by line error checking, translation and execution. Slower in execution compared to compilers Interpreters are mainly used for training purpose and initial stage of development, for easily locating errors. But once the development is completed, compiler can be used. Eg: BASIC		
2	**File Operations** A file is an **abstract data type.** The operating system can provide system calls to create, write, read, reposition, delete, and truncate files. • **Creating a file.** • Two steps are necessary to create a file. • First, space in the file system must be found for the file. • Second, an entry for the new file must be made in the directory. • **Writing a file.** • To write a file, we make a system call specifying both the name of the file and the information to be written to the file. • 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. • Because a process is usually either reading from or writing	6 x 1	6

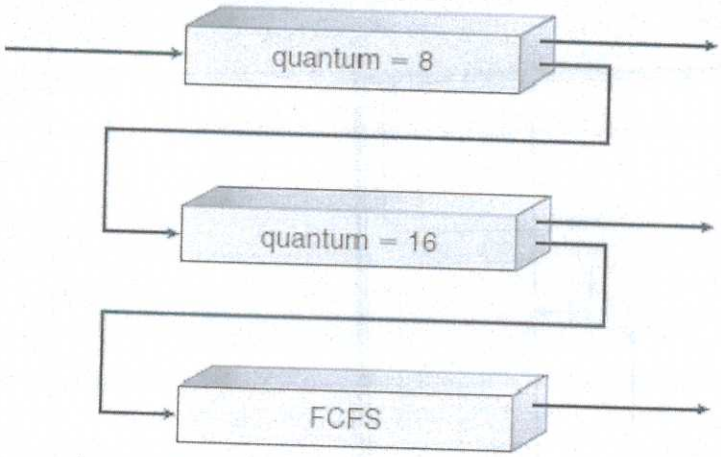
	to a file, the current operation location can be kept as a per-process current file-position pointer. • Both the read and write operations use this same pointer, saving space and reducing system complexity. • **Repositioning within a file.** • The directory is searched for the appropriate entry, and the current-file-position pointer is repositioned to a given value. • Repositioning within a file need not involve any actual I/O. This file operation is also known as a file seek. • **Deleting a file.** • To delete a file, we search the directory for the named file. • Having found the associated directory entry, we release all file space, so that it can be reused by other files, and erase the directory entry. • **Truncating a file.** • The user may want to erase the contents of a file but keep its attributes. • Rather than forcing the user to delete the file and then recreate it, this function allows all attributes to remain unchanged—except for file length—but lets the file be reset to length zero and its file space released.		
3	**Process Control Block, PCB** Each process is represented in the operating system by a process control block (*PCB)* also called a task control block. PCB is the data structure used by the operating system. From the creation to termination of a process, PCB controls all activities related to that process like context switching, execution, maintaining interim register values, keep track of process state, open files and I/O devices, event logging etc. The PCB structure given below facilitates these functions	2 4	6

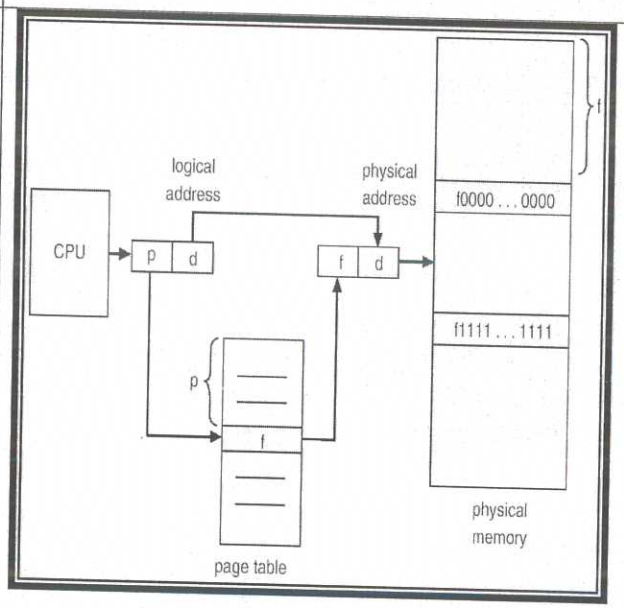


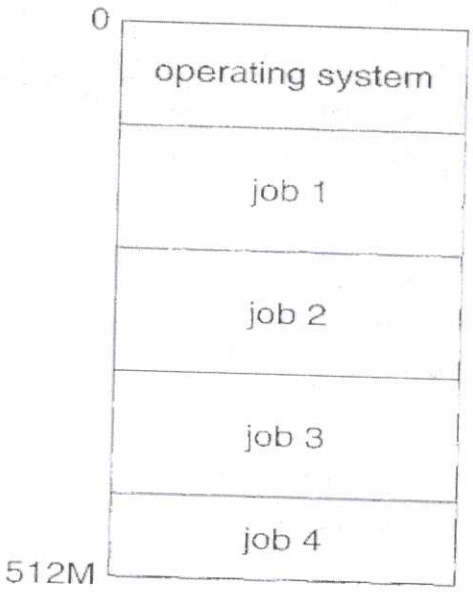
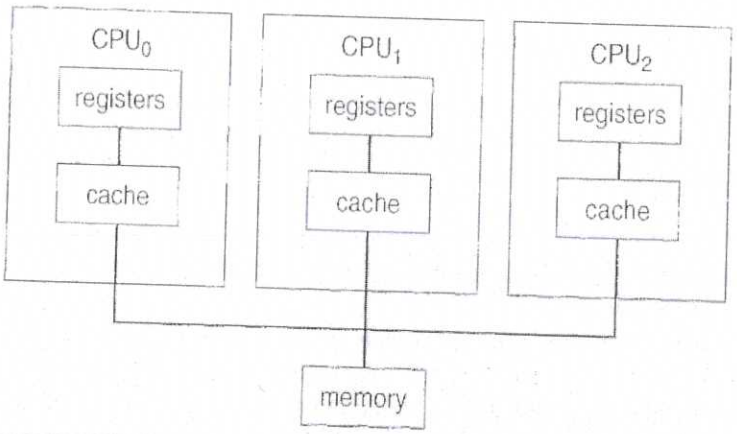
Process state. Shows the process state such as new, ready, running, waiting, halted, and so on.
Program counter. The counter indicates the address of the next instruction to be executed for this process.
CPU registers. Used in context switching

CPU-scheduling information. This information includes a process priority, **pointers** to scheduling queues, and any other scheduling parameters.
Memory-management information. This information may include such information as the value of the base and limit registers, the page tables,or the segment tables, depending on the memory system used by the operating system
Accounting information. This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.
I/O status information. This information includes the list of I/O devices allocated to the process, a list of open files, and so on. Useful in context switching

4	→ The multilevel feedback queue scheduling algorithm partitions the ready queue into several separate queues and allows a process to move between queues. → The idea is to separate processes according to the characteristics of their CPU bursts. → If a process uses too much CPU time, it will be moved to a lower-priority queue. → This scheme leaves I/O-bound and interactive processes in the higher-priority queues. → In addition, a process that waits too long in a lower-priority queue	Fig : 2 Explanation : 4	6

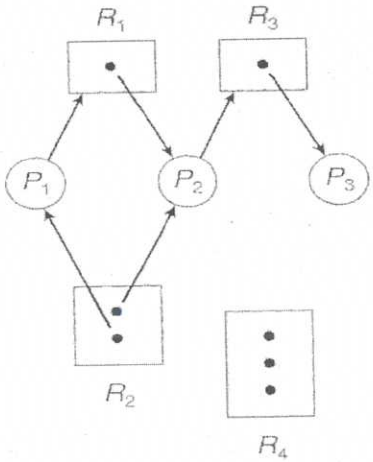
	may be moved to a higher-priority queue. → This form of aging prevents starvation.  → In general, a multilevel feedback queue scheduler is defined by the following parameters: ○ The number of queues ○ The scheduling algorithm for each queue ○ The method used to determine when to upgrade a process to a higher priority queue ○ The method used to determine when to demote a process to a lower priority queue ○ The method used to determine which queue a process will enter when that process needs service.		
5	The binding of instructions and data to memory addresses can be done at the following steps: • **Compile time**: If it is known at compile time where the process will reside in memory, then absolute code can be generated. For example, if it is known that a user process resides 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. • **Load time**: If it is ***not*** known at compile time where the process will reside in memory, then the compiler must generate relocatable code. In this case, final binding is delayed until load time. If the starting address changes, we need only to 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	2 m x 3	6

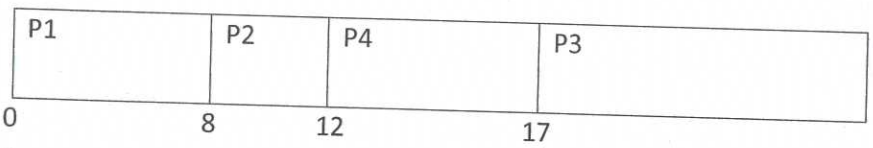
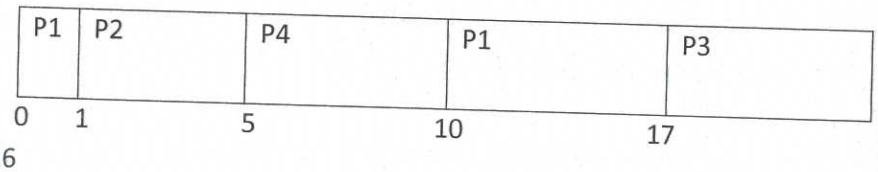
	delayed until run time. Special hardware must be available for this scheme to work. Most general-purpose operating systems use this method		
6	 Every address generated by the CPU is divided into two parts: a page number **(p)** and a page offset **(d).** The page number is used as an index into a page table. The page table contains the base address of each page in physical memory. This base address is combined with the page offset to define the physical memory address that is sent to the memory unit.	Fig : 3 Explain : 3	
7	**Need of virtualization** ✓ Virtualization can help companies maximize the value of IT investments, decreasing the server hardware footprint, energy consumption, and cost and complexity of managing IT systems while increasing the flexibility of the overall environment. Virtualization can affect • Cost • Administration • Fast Deployment • Reduced Infrastructure Costs Explain each	1 ½ x 4	6

III a	In multiprogramming OS several jobs resides in main memory. One job is brought to CPU and executes, and when this job has to wait for some task, like an I/O operation, OS switches to another job in memory and executes it in CPU, and so on. CPU utilization increases  In multiprocessing OS, OS supports execution of multiple processes in multiple processors or different portions of a single process in multiple processors. Presence of multiple processors can increase the addressable memory  	2 ½ 1 ½ 2 ½ 1 ½	
b	Assembler is used to translate the assembly language program in to machine language so that system can understand and execute it. Easier to understand, easy to correct and modify program instructions, same efficiency as machine language.- machine dependent Assembler's functions • Convert mnemonic operation codes to their machine language equivalents		

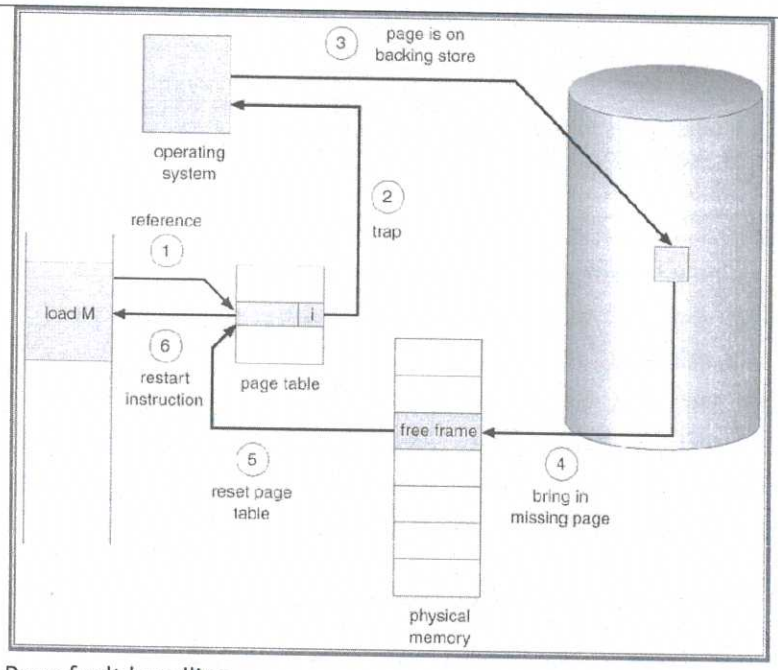
	• Convert symbolic operands to their equivalent machine addresses • Build the machine instructions in the proper format • Convert the data constants to internal machine representations • Write the object program and the assembly listing	7	
IV a	Batch processing – The users of batch operating system do not interact with the computer directly. To speed up processing, jobs with similar needs are batched together and run as a group. The problems with Batch Systems are following. Lack of interaction between the user and job. CPU is often idle, because the speeds of the mechanical I/O devices is slower than CPU. Difficult to provide the desired priority. Time sharing – Give time slice or quantum to each process there by making an illusion that each process is served by the processor at all time. Time-sharing is a logical extension of multiprogramming with an objective to minimize response time. Multiple jobs are executed by the CPU by switching between them, but the switches occur so frequently. Advantages : Quick response, Avoids duplication of software, Reduces CPU idle time	4 4	8
b	Loader The program that brings the program into memory for execution is called a loader. It is a system program which takes the object code of a program as input and prepares it for execution. Functions **Allocation**: The loader determines and allocates the Required memory space for the program to execute properly. **Linking:** Analyses & resolve the symbolic reference made in the object modules **Relocation:** The loader maps & relocates the address references to correspond to the newly allocated memory space during execution. **Loading:** The loader actually loads the machine code corresponding to the object modules into the allocated memory space & makes the program ready to execute.	3 4	7
V a	A process can give up CPU in two ways **Non-preemptive:** A scheduling discipline is non-preemptive if, once a process has been given the CPU, the CPU cannot be taken away from that process. Process can not be	2 ½	5

	interrupted till it terminates or switches to waiting state. **Preemptive**: A scheduling discipline is pre-emptive if, once a process has been given the CPU can be taken away. Process can be interrupted in between.	2 ½	
b	A set of process is in a deadlock state if each process in the set is waiting for an event that can be caused by only another process in the set. The necessary and sufficient conditions for a deadlock to occur are 1.Mutual Exclusion – only one process at a time can use the resource. Other process has to wait. 2. Hold & Wait – holding one resource and waiting to acquire additional resource that is currently held by other process. 3. No pre-emption – A resource can be released only after the process holding it has completed its task. 4. Circular wait – The processes in the system form a circular list or chain where each process in the list is waiting for a resource held by the next process in the list. **Resource-Allocation Graph** Deadlocks can be described more precisely in terms of a directed graph called a **Resource-Allocation Graph.** This graph consists of a set of vertices *V* and a set of edges *E.* The set of vertices Vis partitioned into two different types of nodes: $P == \{P_1, P2, ..., Pn\}$, the set consisting of all the active processes in the system, and $R == \{R_1, R2, ..., RmL$ the set consisting of all resource types in the system. A directed edge from process g to resource type *Rj* is denoted by $P_i \rightarrow R_j$; it signifies that process *Pi* has requested an instance of resource type *Rj* and is currently waiting for that resource. A directed edge from resource type *Rj* to process P_i; is denoted by $R_j \rightarrow P_i$; it signifies that an instance of resource type *Rj* has been allocated to process P_i;. A directed edge $P_i \rightarrow R_j$ is called a request edge; a directed edge $R_j \rightarrow P_i$ is called an assignment edge. The resource-allocation graph shown in Figure below depicts the following situation. The sets *P,* K and *E:* $P == \{P1, P2, P3\}$ $R == \{R1, R2, R3, R4\}$ $E == \{P_1 \rightarrow R_1, R_1 \rightarrow P_2, P_2 \rightarrow R_3, R_2 \rightarrow P_2, R_2 \rightarrow P_1, R_3 \rightarrow P_3\}$	4 x 1 6	10

	Resource instances: o One instance of resource type R_1 o Two instances of resource type R_2 o One instance of resource type R_3 o Three instances of resource type R_4 Process states: o Process P_1 is holding an instance of resource type R_2 and is waiting for an instance of resource type R_1. o Process P_2 is holding an instance of R_1 and an instance of R_2 and is waiting for an instance of R_3. o Process P_3 is holding an instance of R_3.  Figure 7.2 Resource-allocation graph. If each resource type has exactly one instance, then a cycle implies that a deadlock has occurred. If the cycle involves only a set of resource types, each of which has only a single instance, then a deadlock has occurred. Each process involved in the cycle is deadlocked. In this case, a cycle in the graph is both a necessary and a sufficient condition for the existence of deadlock. If each resource type has several instances, then a cycle does not necessarily imply that a deadlock has occurred. In this case, a cycle in. the graph is a necessary but not a sufficient condition for the existence of deadlock.		
VI a	**Critical section** is a segment of code in which the process may be changing common variables, updating tables, writing files so on. **Critical-section problem** is to design a protocol that the process can use to cooperate. -each process must request permission to enter its critical section. - entry section is the section of code implement this request. - exit section follows entry section. - remainder section contains remaining code.	2	5

	Do { [entry section] critical section [exit section] remainder section } while (true); A solution to the critical-section problem must satisfy the following three requirements: 1. **Mutual exclusion.** If process *Pi* 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 deciding 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	3	
b	SJF  Waiting time P1 =0, Waiting time P2 =7, Waiting time P3 =(17-2)=15 Waiting time P4 =(12-3)=9 Average waiting time = (0+7+15+9)/4=31/4 = SRTN  Waiting time P1 =0+(10-1)=9, Waiting time P2 =0, Waiting time P3 =17-	2 ½ 2 2 ½	10

	3=14, Waiting time P4 =5-3=2 Average waiting time = (9+0+14+2)/4=25/4 SRTN is optimal as far as average waiting time is concerned	2 1	
VII a	• **Firstfit** : Allocate the first hole that is big enough. Searching can start either at the beginning of the set of holes or where the previous first-fit search ended. Searching is stopped as soon as a free hole that is large enough is found. • **Bestfit** : Allocate the smallest hole that is big enough. The entire list is to be searched, unless the list is kept ordered by size. This strategy produces the smallest leftover hole. • **Worstfit** : Allocate the largest hole. Here also, the entire list is to be searched, 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-fit approach.	2 m x 3	6
b	The technique to load pages only as they are needed is known as demand paging and is commonly used in virtual memory systems. A demand-paging system is similar to a paging system with swapping where processes reside in secondary memory The procedure for handling page fault is as follows: 1. An internal table (usually kept with the process control block) is maintained for this process, to determine whether the reference was a valid or invalid memory access. 2. If the reference was invalid, the process is terminated. If it was valid, but yet not brought in that page, that page is brought in. 3. A free frame is found. 4. A disk operation is scheduled to read the desired page into the newly allocated frame. 5. When the disk read is complete, the internal table kept with the process and the page table are modified to indicate that the page is now in memory. 6. The instruction that was interrupted by the illegal address trap is restarted.	2 4 ½	9



Page fault handling

2 ½

VIII a

Optimal PRA

5

10

7	0	2	~~0~~ 3	~~0~~ 4	~~2 3~~ 0	~~3 2 2 0~~ 7	~~0~~ 1
7	7	7	3	3	3	3	1
	0	0	0	4	0	0	0
		2	2	2	2	7	7

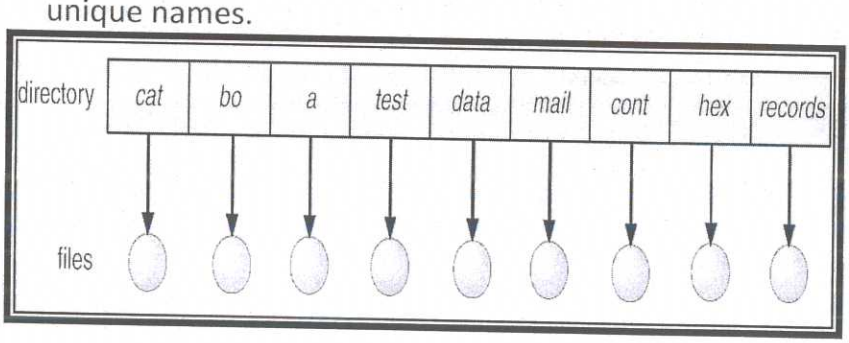
Page faults =8

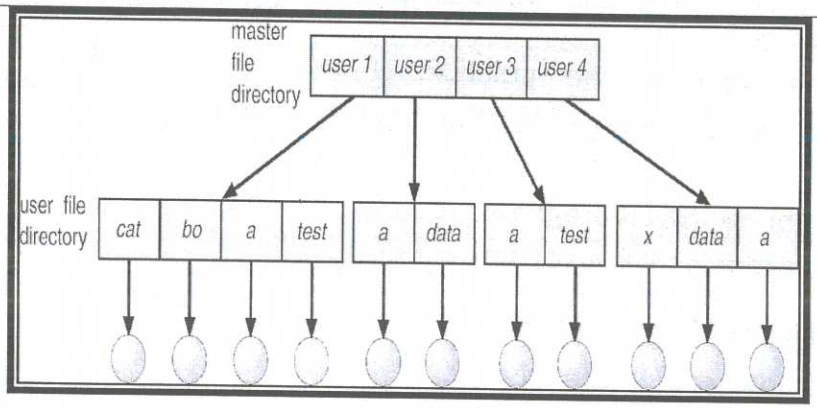
LRU PRA

5

7	0	2	~~0~~ 3	~~0~~ 4	2	3	0	~~3 2 2 0~~ 7	~~0~~ 1
7	7	7	3	3	2	2	2	2	1
	0	0	0	0	0	3	3	7	7
		2	2	4	4	4	0	0	0

Page faults =10

b	**External fragmentation** **Internal fragmentation**	Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous so it can not be used. Memory block assigned to process is bigger. Some portion of memory is left unused as it can not be used by another process.	2 x 2 ½	5
IX a	**Single-Level Directory** • The simplest directory structure is the single-level directory. • All files are contained in the same directory, which is easy to support and understand • A single-level directory has significant limitations, however, when the number of files increases or when the system has more than one user. • Since all files are in the same directory, they must have unique names.  **Two-Level Directory** • A single-level directory often leads to confusion of file names among different users. • The standard solution is to create a separate directory for each user. • In the two-level directory structure, each user has his own **user file directory (UFD).** • The UFDs have similar structures, but each lists only the files of a single user. • When a user job starts or a user logs in, the system's **master file directory (MFD)** is searched. • The MFD is indexed by user name or account number, and each entry points to the UFD for that user. • When a user refers to a particular file, only his own UFD is searched. • Thus, different users may have files with the same name, as long as all the file names within each UFD are unique.	2 1 2	9



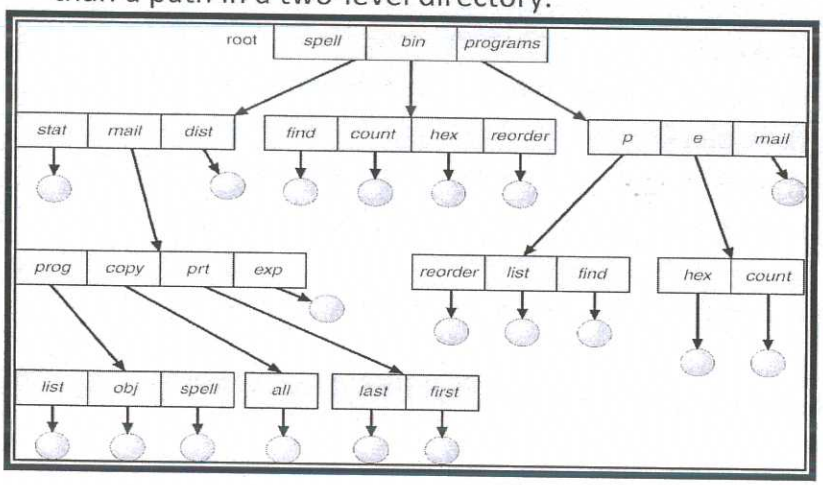
1

Tree-Structured Directories

- 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 (or subdirectory) contains a set of files or subdirectories.
- A directory is simply another file, but it is treated in a special way.
- All directories have the same internal format. One bit in each directory entry defines the entry as a file (0) or as a subdirectory (1).

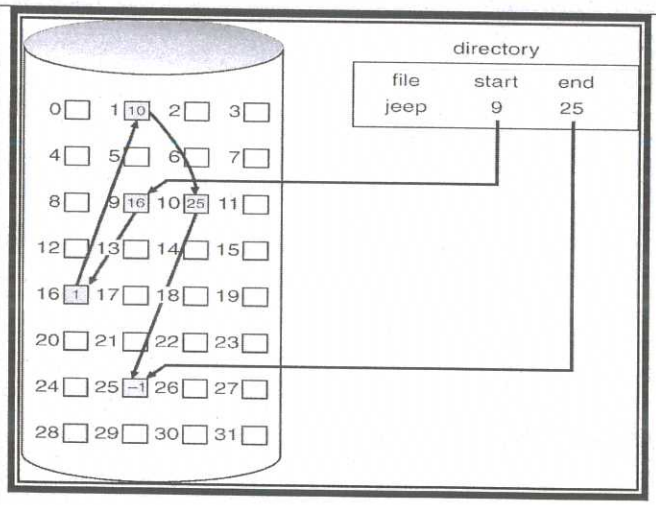
2

- Special system calls are used to create and delete directories.
- With a tree-structured directory system, users can be allowed to access, in addition to their files, the files of other users.
- A path to a file in a tree-structured directory can be longer than a path in a two-level directory.



1

b	**Software Virtualization** ✓ Software virtualization is the practice of running software from a remote server rather than on the user's computer. ✓ Dynamic link library (DLL) programs redirect all the virtualized application's calls to the server's file system. **Advantages of Software Virtualization** 1) Client Deployments Become Easier 2) Easy to manage 3) Software Migration is easier	3 3	6
X a	**Linked 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. • Each block contains a pointer to the next block. These pointers are not made available to the user. • To create a new file, we simply create a new entry in the directory. • With linked allocation, each directory entry has a pointer to the first disk block of the file. • This pointer is initialized to nil (the end-of-list pointer value) to signify an empty file. The size field is also set to 0. • A write to the file causes the free-space management system to find a free block, and this new block is written to and is linked to the end of the file. • To read a file, we simply read blocks by following the pointers from block to block. • There is no external fragmentation with linked allocation, and any free block on the free-space list can be used to satisfy a request. • The size of a file need not be declared when that file is created. • A file can continue to grow as long as free blocks are available.	Explanation : 2 ½ Fig : 2	9

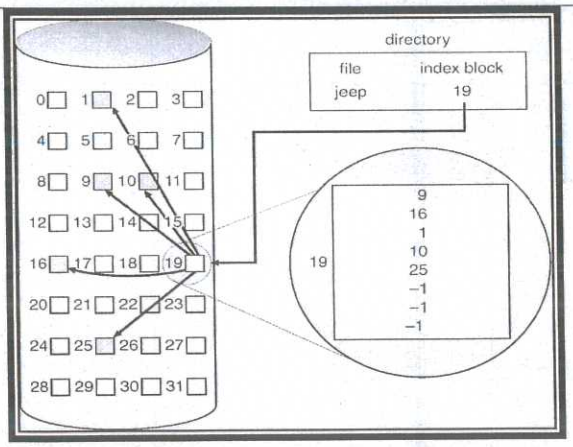


Indexed Allocation

- 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 j^{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 j^{th} index-block entry.
- When the file is created, all pointers in the index block are set to nil.
- When the i^{th} block is first written, a block is obtained from the free-space manager, and its address is put in the z^{th} index-block entry.
- Indexed allocation supports direct access, without suffering from external fragmentation, because any free block on the disk can satisfy a request for more space.
- Indexed allocation does suffer from wasted space, however.

Explanation : 2 ½

Fig : 2

			
b	• **Full Virtualization** ○ In full virtualization, the guest operating system is unaware that it is in a virtualized environment ○ The hardware is virtualized by the host operating system ○ The guest can issue commands to what it thinks is actual hardware, but really are just simulated hardware devices created by the host. • **Para Virtualization** ○ In para virtualization the guest operating system is aware that it is a guest and it has drivers for it ○ Instead of issuing hardware commands, simply issues commands directly to the host operating system. • **Partial Virtualization** ○ The virtual machine simulates multiple instances of an underlying hardware environment. ○ The entire operating systems cannot run in the virtual machine but many applications can run. ○ This type of virtualization is far easier to execute than full virtualization.	2 2 2	6