SCHEME OF VALUATION

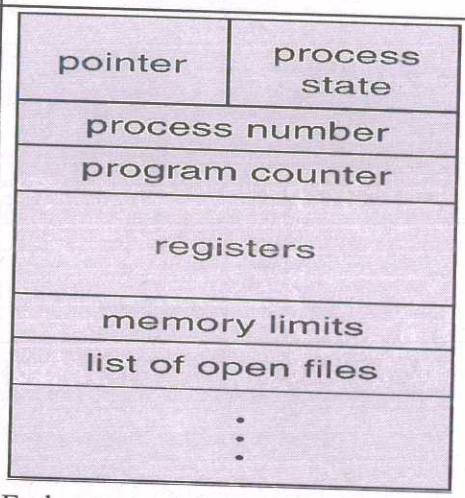
Scoring Indicators

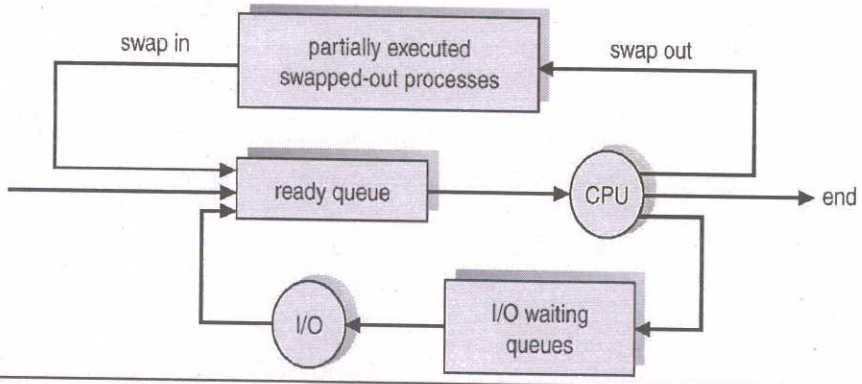
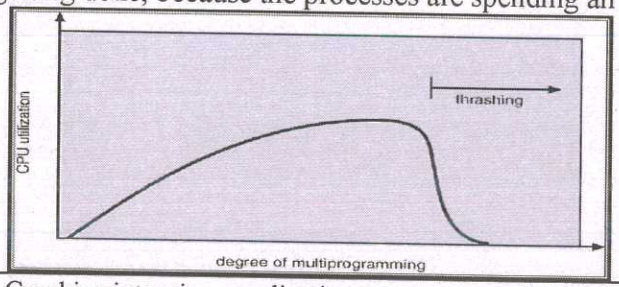
Revision: 2015 **Course Code:4134**

Course Title: OPERATING SYSTEMS

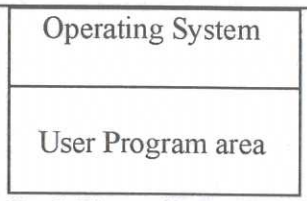
Qst No	Scoring Indicators	Split up score	Sub Total	Total
	Part -A			
I 1)	The goal is to maximize **ease of use**		2	10
2)	If several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place is called race condition.		2	
3)	Physical address binding and logical address binding		2	
4)	External fragmentation exists when enough total memory space exists to satisfy a request, but it is not contiguous.		2	
5)	Operating system virtualization is the use of software to allow system hardware to run multiple instances of different operating systems concurrently		2	
	Part -B			
II 1)	Multiprogramming increases CPU utilisation by organizing jobs so that the CPU always has one to execute. The OS keeps several jobs in memory. The set of jobs is a subset of the jobs kept in the **job pool**. All the jobs that enter the system are kept in the job pool. If several jobs are ready to be brought into memory, and there is not enough room for all of them, the system chooses among them, this is called **job scheduling**. If several jobs are ready to run at the same time, in memory, the system must choose among them to assign the processor, this is called **CPU scheduling.**	**Fig 2 + Expl 4**	6	

OS
Job 1
Job 2
Job 3
Job 4

2)	• Linux is free of cost whereas Windows is costly • Linux software is open source whereas windows is not • Linux is more secure whereas windows is vulnerable to attack by viruses and other malware • Linux has either primary or logical partition, windows has only primary partition • Linux uses backslash for separation of directories, whereas windows uses forward slash • Filenames in Linux are case sensitive whereas filenames in windows are not • The file system in Linux is EXT2, EXT3, EXT4, whereas Windows is FAT, FAT32, NTFS • The types of kernel used in Linux is monolithic whereas windows uses microkernel • The efficiency of running windows is less than linux	**Any 6 points**	**6**	**30**
3)	 Each process is represented in the operating system by a process control block(PCB) also called a task control block. It contains many pieces of information associated with a specific process, including these: **Process state:** The state may be new, ready, running, waiting or halted. **Program counter:** The counter indicates the address of the next instruction to be executed for this process. **CPU registers:** They include accumulators, index registers, stack pointers, and general-purpose registers **CPU-scheduling information:** This information includes a process priority, pointers to scheduling queues, and other scheduling parameters. **Memory-management information:** This information may include the value of the base and limit registers, the page tables, or the segment tables. **Accounting information:** This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers. **I/O status information:** This information includes the list of I/O devices allocated to the process, a list of open files.	**Diag 3 + Points 3**	**6**	

4)	**Long-term scheduler (or job scheduler)** selects which processes should be brought into the ready queue. Long-term scheduler is invoked very infrequently (seconds). It controls the degree of multiprogramming. **Short-term scheduler (or CPU scheduler)** selects which process should be executed next and allocates CPU. It is invoked very frequently (milliseconds) and must be fast. **Medium-term scheduler** removes a process from memory and thus reduce the degree of multiprogramming. Later, the process can be reintroduced into memory, and its execution can be continued where it left off. This scheme is called **swapping**. The process is swapped out, and is later swapped in, by the medium-term scheduler. 	**2 *3**	**6**	
5)	A process is thrashing if it is spending more time paging than executing. If CPU utilization is too low, the degree of multiprogramming is increased by introducing a new processto the system. A global page-replacement algorithm is used; it replaces pages with no regard to the process to which they belong. If a process enters a new phase in its execution, it will need more frames. It starts faulting and taking frames away from other processes. These processes need those pages, and so they also fault, taking frames from other processes. These faulting processes must use the paging device to swap pages in and out. As they queue up for the paging device, the ready queue empties. As processes wait for the paging device, CPU utilization decreases.The CPU scheduler sees the decreasing CPU utilization, and increases the degree of multiprogramming as a result. The new process tries to get started by taking frames from running processes, causing more page faults. No work is getting done, because the processes are spending all their time paging. 	**4 + 2 (fig)**	**6**	
6)	• Graphics-intensive applications are not well suited for today's virtual environment. Video cards cannot handle the requirements of a high-performance graphics adapter. • Gaming, CAD, and software requiring three-dimensional graphics are not ideal for a virtualized environment.			

	• Databases and business intelligence software are also poor matches for virtualization, simply because they require a lot more memory and processor power than current virtualized servers can provide. • Server applications that require access to hardware like PCI cards and USB devices are difficult to virtualize. • Server virtualization doesn't typically play well with proprietary hardware.		6	
7)	a) Creating a file b) Writing a file c) Reading a file d) Repositioning within a file e) Deleting a file f) Truncating a file		6	
	Part -C			
III a)	High reliability, scalability and powerful features make UNIX a popular operating system. Multitasking and Portability - The main features of UNIX include multiuser, multitasking and portability capabilities. Multiple users access the system by connecting to points known as terminals. Several users can run multiple programs or processes simultaneously on one system. UNIX uses a high-level language that is easy to comprehend, modify and transfer to other machines, which means you can change language codes according to the requirements of new hardware on your computer. The Kernel and the Shell - The hub of a UNIX operating system, the kernel manages the applications and peripherals on a system. Together, the kernel and the shell carry out your requests and commands. You communicate with your system through the UNIX shell, which translates to the kernel. When you turn on your terminal, a system process starts that overlooks your inputs. When you enter your password, the system associates the shell program with your terminal. The UNIX shell is a program that gives and displays your prompts and, in conjunction with the kernel, executes your commands. The shell even maintains a history of the commands you enter, allowing you to reuse a command by scrolling through your history of commands. Files and Processes - All the functions in UNIX involve either a file or a process. Processes are executions of programs, while files are collections of data created by you. Files may include a document, programming instructions for the system or a directory. UNIX uses a hierarchical file structure in its design that starts with a root directory--signified by the forward slash (/). The root is followed by its subdirectories, as in an inverted tree, and ends with the file.		7	
III b)	**Batch system** – The common input devices were card readers and tape drives. The common output devices were line printers, tape drives and card punches. The user prepared a job consisting of the program, data and control information and submitted it to the computer operator. The job is in the form of punch cards and the output appeared at a later time. The output consisted of the result of the program, the dump of final memory and register contents for debugging. To speed up processing, the operators batched together jobs with similar needs and ran them through the computer as a group.			**15**

	 **Real-Time System**- 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. **Hard real-time**: It requires that all delays in the system be bounded and critical tasks be completed on time. Secondary storage limited or absent, data stored in short term memory, or read-only memory (ROM). **Soft real-time**: Critical task gets priority over other tasks and retains priority until it is completed. Limited utility in industrial control of robotics. Useful in applications like multimedia, virtual reality etc requiring advanced operating system features.	**4 + 4**	**8**	
IV a)	**Assembler** – The Assembler is used to translate the program written in Assembly language into machine code. The source program is a input of assembler that contains assembly language instructions. The output generated by assembler is the object code or machine code understandable by the computer. **Compiler** – The language processor that reads the complete source program written in high level language as a whole in one go and translates it into an equivalent program in machine language is called as a Compiler. Example: C, C++, C#, Java. In a compiler, the source code is translated to object code successfully if it is free of errors. The compiler specifies the errors at the end of compilation with line numbers when there are any errors in the source code. The errors must be removed before the compiler can successfully re-compile the source code again. **Interpreter** – The translation of single statement of source program into machine code is done by language processor and executes it immediately before moving on to the next line is called an interpreter. If there is an error in the statement, the interpreter terminates its translating process at that statement and displays an error message. The interpreter moves on to the next line for execution only after removal of the error. An Interpreter directly executes instructions written in a programming or scripting language without previously converting them to an object code or machine code. Example: Perl, Python and Matlab.	**1 + 3 + 3**	**7**	**15**
IVb)	**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 deallocate memory space as needed.	**4 + 4**	**8**	

	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.			

V a)	**The Critical-Section Problem** – There are n processes all competing to use some shared data. Each process has a code segment, called critical section, in which the shared data is accessed. Problem – ensure that when one process is executing in its critical section, no other process is allowed to execute in its critical section.
	Solution to Critical-Section Problem should satisfy
	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 there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely.
	3. **Bounded Waiting**. A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.
	2 process solution - Only 2 processes, P0 and P1
	General structure of process Pi (other process Pj)
	do { entry section
	critical section
	exit section
	reminder section
	} while (1);
	Processes may share some common variables to synchronize their actions.
	Algorithm 1 Shared variables: int turn;
	initially turn = 0, when turn = i, Pi can enter its critical section
	Process Pi
	do { while (turn != i) ;
	critical section
	turn = j;
	reminder section
	} while (1);
	Satisfies mutual exclusion, but not progress
	Algorithm 2 Shared variables boolean flag[2]; initially flag [0] = flag [1] = false. When flag [i] = true, Pi ready to enter its critical section
	Process Pi
	do { flag[i] := true;
	while (flag[j]);
	critical section
	flag [i] = false;
	remainder section
		3 + 2*3 algs	**9**	

	} while (1);
	Satisfies mutual exclusion, but not progress requirement.
	Algorithm 3 – Combined shared variables of algorithms 1 and 2.
	Process Pi
	do {
	flag [i]:= true;
	turn = j;
	while (flag [j] and turn = j) ;
	critical section
	flag [i] = false;
	remainder section
	} while (1);
	Meets all three requirements; solves the critical-section problem for two processes.
				15

V b)	Deadlock – A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set. Deadlock can arise if four conditions hold simultaneously ■ Mutual exclusion: only one process at a time can use a resource. ■ Hold and wait: a process holding at least one resource is waiting to acquire additional resources held by other processes. ■ No preemption: a resource can be released only voluntarily by the process holding it, after that process has completed its task. ■ Circular wait: there exists a set {P0, P1, …, Pn} of waiting processes such that P0 is waiting for a resource that is held by P1, P1 is waiting for a resource that is held by P2, …, Pn–1 is waiting for a resource that is held by Pn, and Pn is waiting for a resource that is held by P0.	**2 + 1*4 pts**	**6**	
VI a)	**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 cannot be interrupted till it terminates or switches to waiting state. Preemptive: A scheduling discipline is preemptive if, once a process has been given the CPU can taken away. Process can be interrupted in between. CPU scheduling decisions may take place when a process: • Switches from running to waiting state. • Switches from running to ready state. • Switches from waiting to ready. • Terminates. Scheduling under 1 and 4 is non-preemptive. All other scheduling is preemptive.		**3**	**15**
VI b)	**First-Come, First-Served Scheduling** The simplest CPU-scheduling algorithm is the first-come, first-served (FCFS) scheduling algorithm. With this scheme, the process that requests the CPU first is allocated the CPU first. The implementation of the FCFS policy is easily managed but the average waiting time under the FCFS policy is often quite long. Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds.	**4 m each (2 for eg)**	**12**	

Process	Burst Time
P_1	24
P_2	3
P_3	3

If the processes arrive in the order P1, P2 , P3 , and are served in FCFS order, the Gantt chart, which is a bar chart that illustrates a particular schedule, including the start and finish times of each of the participating processes as follows.

The waiting time is 0 milliseconds for process P1, 24 milliseconds for process P2 and 27 milliseconds for process P3 . Thus, the average waiting time is (0 + 24 + 27)/3 = 17 milliseconds.

The disadvantage of FCFS algorithm is all the other processes wait for the one big process to get off the CPU. This results in lower CPU and device utilization than might be possible if the shorter processes were allowed to go first.

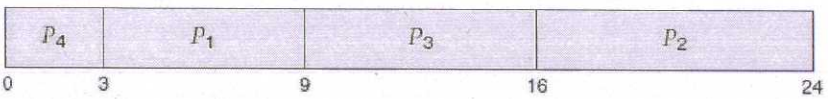
Shortest-Job-First Scheduling

Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time. Two schemes:

- Nonpreemptive – once CPU given to the process it cannot be preempted until completes its CPU burst.
- Preemptive – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is known as the Shortest Remaining Time First (SRTF).

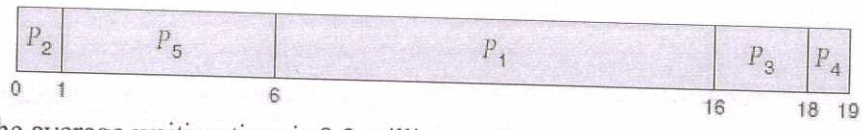
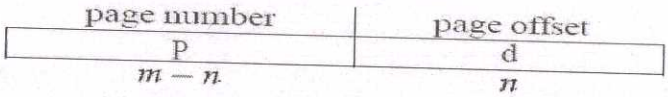
SJF is optimal – gives minimum average waiting time for a given set of processes.

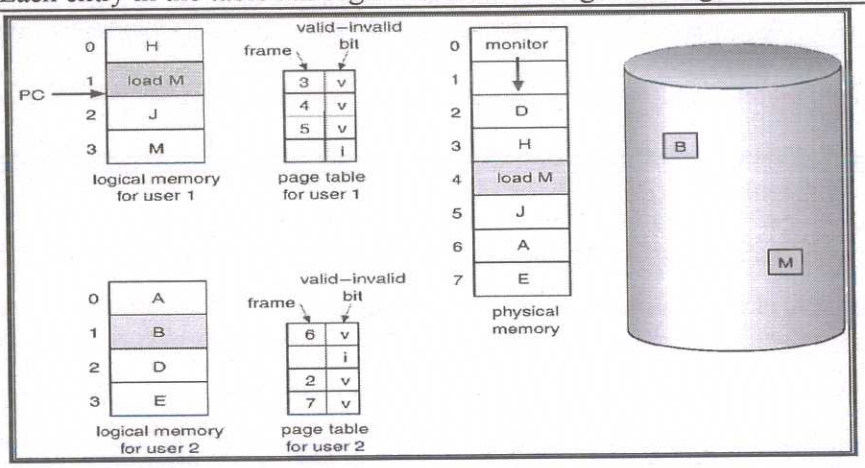
Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

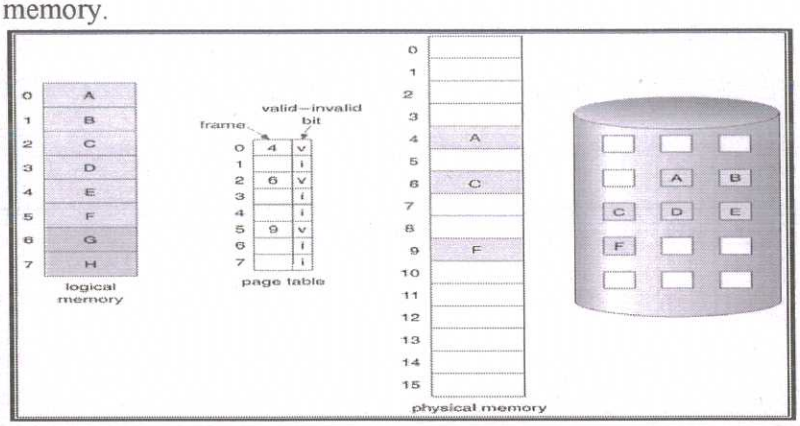
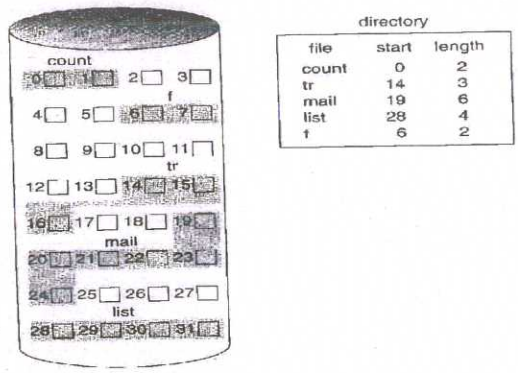


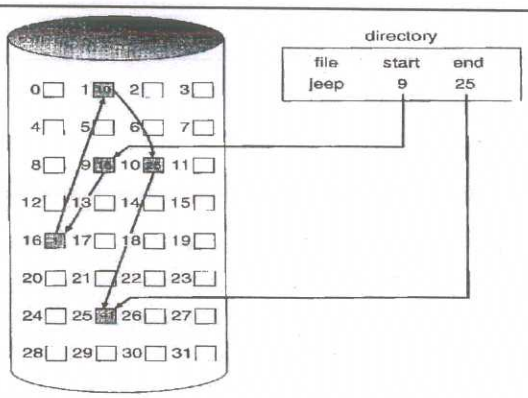
The waiting time is 3 milliseconds for process P1, 16 milliseconds for process P2, 9 milliseconds for process P3, and 0 milliseconds for process P4. Thus, the average waiting time is (3 + 16 + 9 + 0)/4 = 7 milliseconds.

Priority Scheduling - A priority number is associated with each process. The CPU is allocated to the process with the highest priority. Problem is starvation – low priority processes may never execute. A solution is to introduce a variable called Aging – as time progresses increase the priority of the process.

	Process Burst Time Priority P_1 10 3 P_2 1 1 P_3 2 4 P_4 1 5 P_5 5 2  The average waiting time is 8.2 milliseconds.			
VII a)	Paging is a memory-management scheme that permits the physical-address space of a process to be noncontiguous. Support for paging has been handled by hardware. Physical memory is broken into fixed-sized blocks called frames. Logical memory is also broken into blocks of the same size called pages. When a process is to be executed, its pages are loaded into any available memory frames from the backing store. The backing store is divided into fixed-sized blocks that are of the same size as the memory frames. The hardware support for paging is illustrated in Figure. Every address generated by the CPU is divided into two parts: a page number **(p)** and a page offset **(d).** The page number is used as an index into a page table. The page table contains the base address of each page in physical memory. This base address is combined with the page offset to define the physical memory address that is sent to the memory unit. If the size of logical-address space is 2m, and a page size is *2n* addressing units (bytes or words), then the high-order m – *n* bits of a logical address designate the page number, and the *n* low-order bits designate the page offset. Thus, the logical address is as follows  where **p** is an index into the page table and **d** is the displacement within the page. The difference between the user's view of memory and the actual physical memory is reconciled by the **address-translation hardware**. The logical addresses are translated into physical addresses. This mapping is hidden from the user and is controlled by the operating system. Since the operating system is managing physical memory, it must be aware of the allocation details of physical memory: which frames are allocated, which frames are available, how many total frames there are, and so on. This information is generally kept in a data structure called a **frame table**. The frame table has one entry for each physical page frame, indicating whether the latter is free or allocated and, if it is allocated, to which page of which process or processes.	**Expln 6 + 4 (fig)**	**10**	**15**
b)	**Paging:** Transparent to programmer (system allocates memory) No separate protection No separate compiling No shared code It uses page table to map between pages and frames	**1* 5**	**5**	

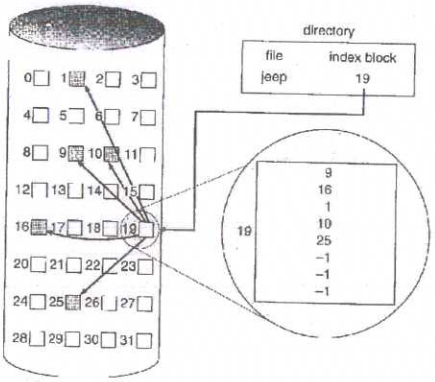
	Each entry in the table has only page number (as pages are of same size) **Segmentation:**Involves programmer (allocates memory to specific function inside code) Separate compiling Separate protection Shares code. It uses segment table to map segments. Each entry in the table has segment name and segment length	**(any 5 pts)**		
VIII a)	 If no frame is free, one frame is found that is not currently being used and free it. This freed frame is used to hold the page for which the process faulted.The routine is: 1. Find the location of the desired page on the disk. 2. Find a free frame: a. If there is a free frame, use it. b. If there is no free frame, use a page-replacement algorithm to select a victim frame. c. Write the victim page to the disk; change the page and frame tables accordingly. 3. Read the desired page into the (newly) free frame; change the page and frame tables. 4. Restart the user process. If no frames are free, two page transfers are required. This situation effectively doubles the page-fault service time and increases the effective access time accordingly. This overhead is reduced by using a **modify bit** (or dirty bit). The modify bit for a page is set by the hardware whenever any word or byte in the page is written into. When a page is selected for replacement, its modify bit is examined. If the bit is set, that page must be written to the disk. If the modify bit is not set, the copy of the page on the disk has not been overwritten.	**Fig 4 + 4 Expln**	**8**	**15**
b)	A demand-paging system is similar to a paging system with *swapping*. Processes reside on secondary memory. When a process is to be executed, it is swapped into memory. When the bit is set to "valid," this value indicates that the associated page is both legal and in memory. If the bit is set to "invalid," this value indicates that the page is valid but is currently on the disk. The page-table entry for a page that is brought into memory is set as usual, but the page-table entry for a page that is not currently in memory is marked invalid, or contains the address of the page on disk.	**Fig 4 + 3 expln**	7	

	If the process executes and accesses pages that are memory resident, execution proceeds normally. But if the process tries to access a page that was not brought into memory, a **page-fault** trap occurs. The paging hardware, in translating the address through the page table, will notice that the invalid bit is set, causing a trap to the operating system. This trap is the result of the operating system's failure to bring the desired page into memory.  			
IX a)	**Contiguous Allocation** – It requires each file to occupy a set of contiguous blocks on the disk. Disk address are linear on a disk, so accessing block b + 1 after block b requires no head movement. Thus the number of disk seeks required for accessing contiguously allocated files is minimal. Contiguous allocation of a file is defined by disk address and length of the block. If a file is n blocks long, and starts at block b, then it occupies block b, b+1, b+2, ... b+n-1. The directory entry of each file indicates the starting block and the length of the area allocated for this file  directory file start length count 0 2 tr 14 3 mail 19 6 list 28 4 f 6 2 This suffers from external fragmentation **Linked Allocation** – Each file is a linked list of disk blocks, the disk blocks may be scattered anywhere on the disk. The directory contains the pointer to the first and last blocks of the file. The directory entry is initialised to nil value to signify an empty file. A write to this file causes a new block to be found, written to and then linked to the end of the file. To read a file, read blocks by following the pointers from block to block.	**3 * 3**	**9**	**15**



The major disadvantage of linked allocation is that it can be used only for sequential access. Another disadvantage is the space required for pointers

Indexed Allocation – It brings all pointers to one location called the index block. Each file has its own index block, which is an array of disk block addresses. The i th entry of the index block points to the i th block of the file.



To read the i the block, we use the i th entry in the index block. Indexed allocation supports direct access without suffering from external fragmentation. It suffers from wasted space due to pointer overhead.

IX b)

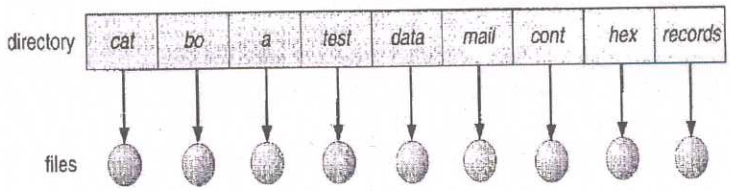
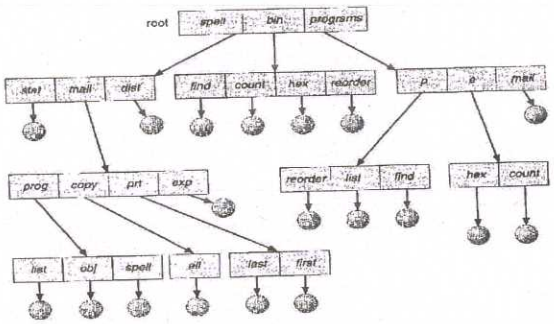
Hardware Virtualization It refers to the creation of a virtual machine that acts like a real computer with an operating system. Software executed on these virtual machines is separated from the underlying hardware resources. Hardware virtualization is further subdivided into the following types:

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

6

	Software Virtualization It 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. The advantages are Client Deployments Become Easier, Easy to manage and Software Migration is easier			
X a)	**Single-Level Directory** - A single directory for all users. All files are in the same directory which is easy to support and understand. Limitations – when the number of files increases or if there is more than one user. It is difficult for file names to be unique in this case.  **Two- level Directory** – Each user has his own User Fie Directory (UFD). Each UFD has a similar structure, but has the files of only one user. When a user logs in the Master File Directory (MFD) is searched, which is listed by the user name for each user. Different users may have files with the same name. The disadvantage is it isolates one user completely from another. **Tree-Structured Directory** – Path names are of 2 types: absolute path name (from the root) and relative path name (from the current directory). It allows a user to define his own sub directories and permits him to impose a structure on his files.  When deleting a directory, MS DOS requires that all subdirectories and files be deleted before a directory is deleted. UNIX rm command deletes all subdirectories and files automatically, when a directory is deleted. In a tree structured directory a user can access, other user's files as well by specifying their absolute path name.	3 *3	9	15
b)	• Support for any standard x86 hardware • Support for a wide variety of Linux and Windows host operating systems, including 64-bit operating systems • Support for a wide variety of Linux, NetWare, Solaris x86, and Windows guest operating systems, including 64-bit operating systems • Support for Virtual SMP enabling a single virtual machine to span multiple physical processors • Quick and easy virtual machine creation with a virtual machine wizard • Virtual machine monitoring and management with a user-friendly remote console	1 * 6 points	6	