SCHEME OF EVALUATION

Scoring Indicators

Revision : 2015 **Course Code : 4134**

Course Title : Operating Systems

Qst. No	Scoring Indicator	Split up Score	Sub Total	Total
I	**PART – A**			
1.	Assembler, Loader, Compiler, Interpreter, OS (Any two)	2	2	
2.	A thread is a basic unit of CPU utilization or a light weight process	2	2	
3.	Internal and External fragmentation	2	2	
4.	A thin client is a lightweight computer that has been optimized for establishing a remote connection with a server-based computing environment	2	2	10
5.	contiguous, linked and indexed allocations	2	2	
	PART – B			
II (1)	COMPILERS It is a tool which translate notations from one system to another, usually from source language to target language. A compiler must handle syntax and semantic errors. Depends on its type, a compiler can generate either assembly language program or machine code from high level language program. INTERPRETERS: It is a translation program which converts each program statement into machine code just before the next program statement is to be executed. Translation and execution occur immediately, one after another, one statement at a time.	6	6	6

Basics for comparison	Compiler	Interpreter
Input	Takes an entire program at a time	Takes single line of code at a time
Output	It generates intermediate object code	It does not produce any intermediate object code
Working Mechanism	Compilation done before execution	Compilation and execution take place simultaneously
Speed	Comparatively faster	Slower
Memory	More memory required	Less memory required
Errors	Displays all errors after Compilation	Displays one error at a time
Error Detection	Difficult	Easier

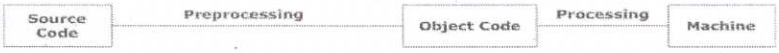


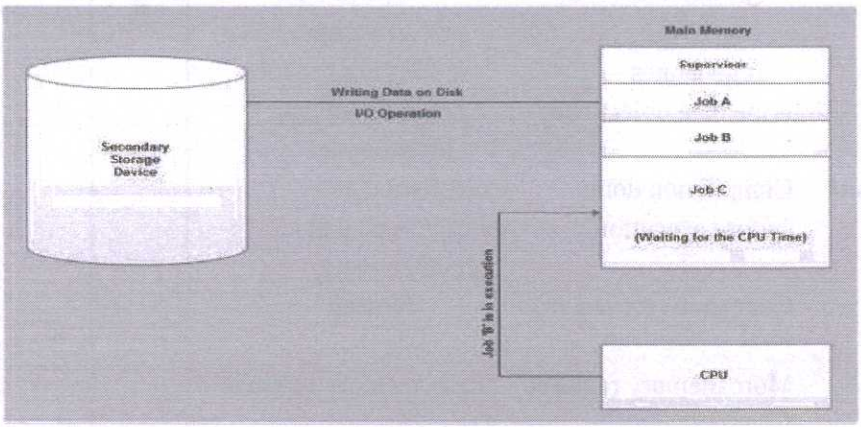
Figure: Compiler

Figure: Interpreter

II (2) MULTIPROGRAMMING SYSTEMS — 3 — **6** — **6**

One of the most important aspects of OS is the ability to multiprogram. A single user cannot keep either CPU or the I/O devices busy all times. Multiprogramming increases CPU utilization by organizing jobs (code and data) so that the CPU always has one to execute.

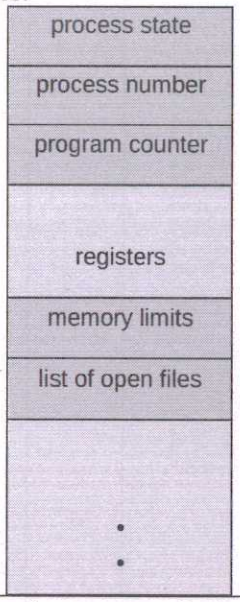
The idea is : The OS keeps several jobs in memory simultaneously. This set of jobs can be a subset of jobs kept in the job pool- which contains all jobs that enter the system- since the number of jobs that can be kept simultaneously in memory is usually smaller than the number of jobs that can be kept in the job pool. The OS picks and begins to execute one of the jobs in memory. Eventually, the job may have to wait for some task, such as an I/O operation, to complete. In a non-multiprogrammed system, the CPU would sit idle. In multiprogrammed system, the OS simply switches to, and executes another job. When that job needs to wait, the CPU is switched to another job, and so on. Eventually, the first job finishes waiting and gets the CPU back. As long as at least one job needs to execute, the CPU is never idle.

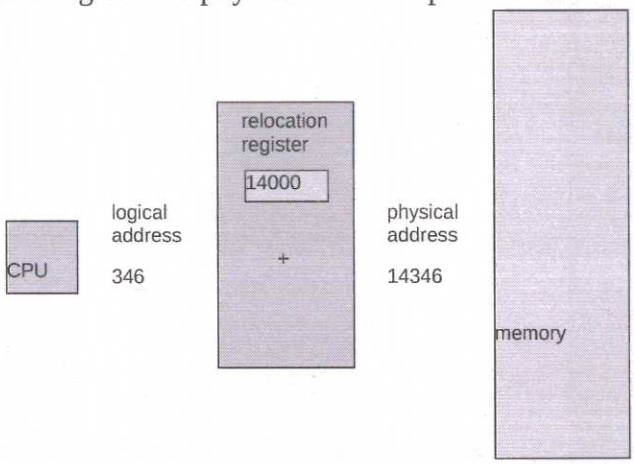


3

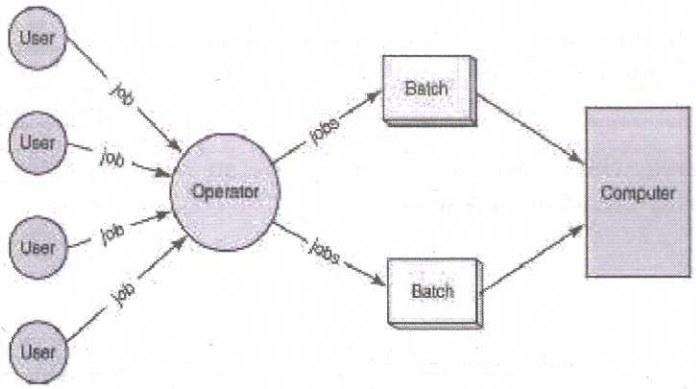
- Advantages:

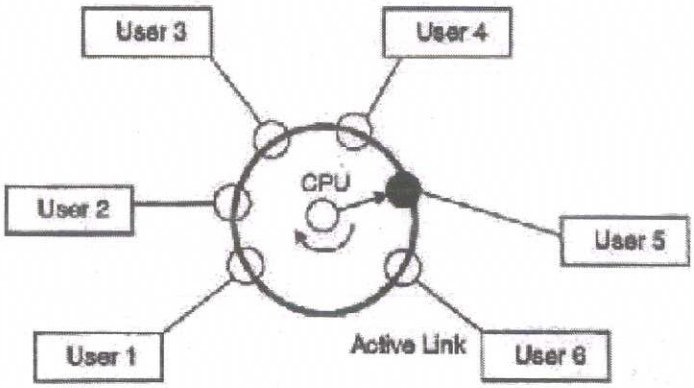
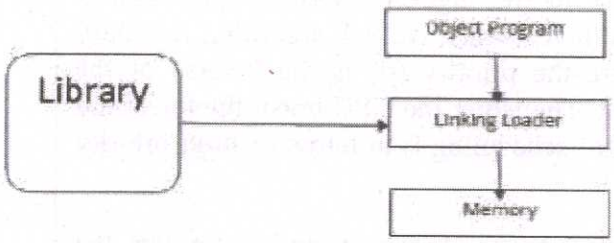
CPU is used most of time and never become idle
The system looks fast as all the tasks runs in parallel
Short time jobs are completed faster than long time jobs
Multiprogramming systems support multiple users
Response time is shorter

	Disadvantages: It is difficult to program a system because of complicated schedule handling Tracking all tasks/processes is sometimes difficult to handle Due to high load of tasks, long time jobs have to wait long			
II (3)	**PROCESS CONTROL BLOCK (PCB)** Each process is represented in the OS by a 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, 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**: It includes a process priority, pointers to scheduling queues, and any other scheduling parameters. • **Memory management information**: It includes information such as value of base and limit registers, the page tables, or the segment tables, depending on the memory system used by the OS. • **Accounting information**: It includes the amount of CPU and real time used, time limits, account members, job or process numbers, and so on. • **I/O status information**: It includes the list of I/O devices allocated to the process, a list of open files, and so on. In brief, PCB serves as the repository for any information that may vary from process to process. 	3 3	6	6
II (4)	A set of processes is in a deadlock state when every process in the set is waiting for an event that can be caused only by another process in the set. A deadlock situation can arise if the following four conditions hold simultaneously in a system: 1. **Mutual exclusion**. At least one resource must be held in a		6	6

	nonsharable mode; that is, 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. 2. **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. 3. **No preemption**. Resources cannot be preempted; that is, a resource can be released only voluntarily by the process holding it, after that process has completed its task. 4. Circular wait. A set P0, P1, ..., Pn of waiting processes must exist such that P0 is waiting for a resource held by P1, P1 is waiting for a resource held by P2, ..., Pn−1 is waiting for a resource held by Pn, and Pn is waiting for a resource held by P0. All four conditions must hold for a deadlock to occur.	**4 X 1.5**		
II (5)	Logical Versus Physical Address Space An address generated by the CPU is commonly referred to as a **logical address**, whereas an address seen by the memory unit—that is, the one loaded into the **memory-address register** of the memory—is commonly referred to as a **physical address**. The compile-time and load-time address-binding methods generate identical logical and physical addresses. However, the execution-time address-binding scheme results in differing logical and physical addresses. In this case, we usually refer to the logical address as a **virtual address**. The set of all logical addresses generated by a program is a **logical address space**. The set of all physical addresses corresponding to these logical addresses is a **physical address space**. Thus, in the execution-time address-binding scheme, the logical and physical address spaces differ.  	**4** **2**	**6**	**6**
II (6)	The general **dynamic storage- allocation problem**, concerns how to satisfy a request of size *n* from a list of free holes. There are many solutions to this problem. The **first-fit**, **best-fit**, and **worst-fit** strategies are the ones most commonly used to select a free hole from the set of available holes.		**6**	**6**

	First fit. Allocate the first hole that is big enough. Searching can start either at the beginning of the set of holes or at the location where the previous first-fit search ended. We can stop searching as soon as we find a free hole that is large enough. **Best fit**. Allocate the smallest hole that is big enough. We must search the entire list, unless the list is ordered by size. This strategy produces the smallest 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-fit approach.	3 3		
II (7)	File is an **abstract data type** • **Create** • **Write** - at **write pointer** location • **Read** - at **read pointer** location • **Reposition within file - seek** • **Delete** • **Truncate** • ***Open(F_i)*** - search the directory structure on disk for entry F_i, and move the content of entry to memory • ***Close (F_i)*** - move the content of entry F_i in memory to directory structure on disk Listing Brief Description……..	 3 3	6	6
	PART – C			
III(a)	• Process Management • Memory Management • Secondary Storage Management • Device Management • File Management • Security and Protection Listing Explanation	 4 4	8	15
III(b)	BASIS FOR COMPARISON / LINUX / WINDOWS Cost: Free of cost / Expensive Open Source: Yes / No Customizable: Yes / No Security: More Secure / Vulnurable to viruses and malware attacks	7	7	

	Booting partition	Either primary or logical partition	Only in primary		
	File Names	Case sensitive	Case insensitive		
	File System	ext2,ext3,ext3 ReisersFS,XFS,JFS	fat,fat32,ntfs ReFS		
	Efficiency	Effective Running efficiency	Lower than Linux		
IV (a)	**BATCH SYSTEMS** A typical computer in the 1960s and 70s was a large machine. Its processing was managed by a human operator. The operator would organize various jobs from multiple users into batches. The central idea is the use of a software known as the **monitor.** With this type of OS, user no longer has direct access to the processor. Instead, the user submits the job on cards or tape to a operator, who batches the jobs together sequentially and places the entire batch on an input device, for use by the monitor. Each program is constructed to branch back to monitor when it completes processing, at which point the monitor automatically begins loading the next program.  Advantages: Repeated jobs are done fast You don't need special hardware and system support to input data in batch systems. Sharing of batch system for multiple users. The idle time batch system is very less. Disadvantages: Computer operators must be trained for using batch systems. It is difficult to debug batch systems. Batch systems are sometime costly. **TIME SHARING SYSTEMS** Multiprogrammed systems provide an environment in which the various system resources are utilized effectively, but they do not provide for user interaction with the computer system. The time sharing (or multitasking) is a logical extension of multiprogramming. In time sharing systems, the CPU executes multiple jobs by switching among them, but the switches occur so frequently that the users can interact with each program while it is	2 2 2	**8**	**15**

	running. Time sharing requires an interactive computer system, which provides direct communication between the user and the system. A time shared OS allows many users to share the computer simultaneously. This system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time-shared computer.  Advantages: Increases performance Many applications can run at the same time Disadvantages: It consumes much resources so it need special operating systems Switching between tasks becomes sometimes sophisticated as there are lot of users and applications running which may hang up the system Time sharing systems should have high specifications of hardware	2		
IV(b)	Loader is a program which accepts the object program decks, prepares this program for execution by the computer and initializes the execution. It is actually responsible for initiating execution of process. Loader must perform four functions: 1. Allocation: Allocate space in memory for the program. 2. Linking: Resolve symbolic references between object decks. 3. Relocation: Adjust all address dependent locations, such as address constants, to correspond to the allocated space. 4. Loading: Physically place the machine instructions and data into  memory.	2 4 1	7	
V (a)	**FIRST-COME,FIRST-SERVED (FCFS) 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 with a FIFO		8	15

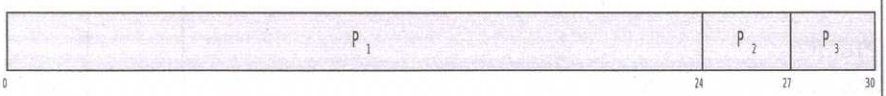
queue. When a process enters the ready queue, its PCB is linked onto the tail of the queue. When the CPU is free, it is allocated to the process at the head of the queue. The running process is then removed from the queue. The code for FCFS scheduling is simple to write and understand.

The average waiting time under the FCFS policy, however, is often quite long. Consider the following set of processes that arrive at time with the length of the CPU burst given in milliseconds

Process	Burst Time
P_1	24
P_2	3
P_3	3

Suppose that the processes arrive in the order: P_1 , P_2 , P_3

The Gantt Chart for the schedule is:



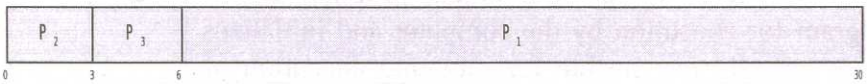
Waiting time for P_1 = 0; P_2 = 24; P_3 = 27

Average waiting time: (0 + 24 + 27)/3 = 17

Suppose that the processes arrive in the order:

P_2 , P_3 , P_1

The Gantt chart for the schedule is:



Waiting time for P_1 = 6; P_2 = 0; P_3 = 3

Average waiting time: (6 + 0 + 3)/3 = 3

Much better than previous case

Convoy effect - short process behind long process

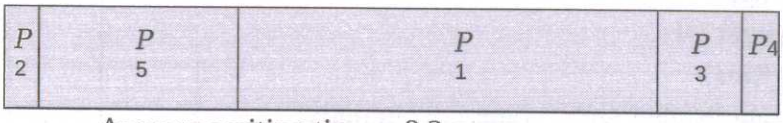
Priority Scheduling

A priority is associated with each process, and the CPU is allocated to the process with the highest priority. Equal-priority processes are scheduled in FCFS order. An SJF algorithm is simply a priority algorithm where the priority (p) is the inverse of the (predicted) next CPU burst. The larger the CPU burst, the lower the priority, and vice versa.The scheduling is in terms of ***high*** priority and ***low*** priority.

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

As an example, consider the following set of processes, assumed to have arrived at time 0 in the order P_1, P_2, , P_5, with the length of the CPU burst given in milliseconds:

Using priority scheduling, we would schedule these processes according to the following Gantt chart:



Average waiting time = 8.2 msec

V (b)

A **cooperating process** is one that can affect or be affected by other processes executing in the system. Cooperating processes can either directly share a logical address space

Concurrent access to shared data may result in data inconsistency

Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes.

Illustration of the problem:

Suppose that we wanted to provide a solution to the consumer-producer problem that fills ***all*** the buffers. We can do so by having an integer **counter** that keeps track of the number of full buffers. Initially, **counter** is set to 0. It is incremented by the producer after it produces a new buffer and is decremented by the consumer after it consumes a buffer.

1.5

Producer

```
while(true){
        /* produce an item in next produced */

        while (counter == BUFFER_SIZE) ;
            /* do nothing */
        buffer[in] = next_produced;
        in = (in + 1) % BUFFER_SIZE;
        counter++;
}
```

1.5

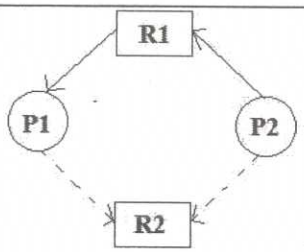
Consumer

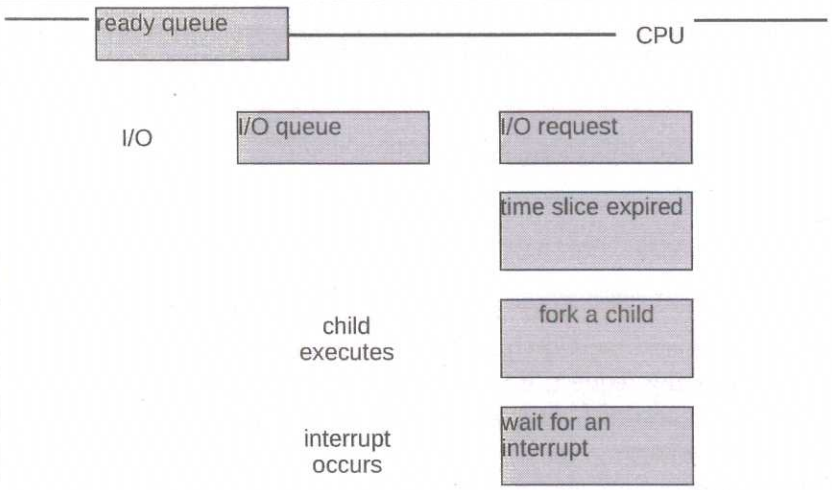
```
while (true) {
        while (counter == 0)
            ; /* do nothing */
        next_consumed = buffer[out];
        out = (out + 1) % BUFFER_SIZE;
      counter--;
        /* consume the item in next consumed */
}
```

1.5

7

	The statement "counter++" may be implemented in machine language (on a typical machine) as follows: **register1=counter** **register1=register1+1** **counter = register1** where *register*$_1$ is one of the local CPU registers. Similarly, the statement "counter--" is implemented as follows: **register2=counter** **register2=register2-1** **counter = register2** Consider this execution interleaving with "count = 5" initially: S0: producer execute **register1 = counter** {register1 = 5} S1: producer execute **register1 = register1 + 1** {register1 = 6} S2: consumer execute **register2 = counter** {register2 = 5} S3: consumer execute **register2 = register2 – 1** {register2 = 4} S4: producer execute **counter = register1** {counter = 6 } S5: consumer execute **counter = register2** {counter = 4} Notice that we have arrived at the incorrect state "counter == 4", indicating that four buffers are full, when, in fact, five buffers are full. If we reversed the order of the statements at T_4 and T_5, we would arrive at the incorrect state "counter == 6". We would arrive at this incorrect state because we allowed both processes to manipulate the variable counter concurrently. A situation like this, where 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 a **race condition**.	**1.5** **1**		
VI (a)	**METHODS FOR HANDLING DEADLOCKS** • **Deadlock Prevention** ➢ Eliminate Mutual Exclusion ➢ Eliminate Hold and wait ➢ Eliminate No Preemption ➢ Eliminate Circular Wait • **Deadlock Avoidence** Try to avoid deadlocks by making use prior knowledge about the usage of resources by processes including resources available, resources allocated, future requests and future releases by processes. Most deadlock avoidance algorithms need every process to tell in advance the maximum number of resources of each type that it may need. Based on all these info we may decide if a process should wait for a resource or not, and thus avoid chances for circular wait. A resource allocation graph is generally used to avoid deadlocks. A claim edge in resource allocation graph denotes that a request may be made in future and is represented as a dashed line. Based on claim edges we can see if there is a chance for a cycle and then grant requests if the systemwill again be in a safe state.In the following RAG If R2 is allocated to p2 and if P1 request for R2, there will be a deadlock.	**2** **2**	**8**	**15**

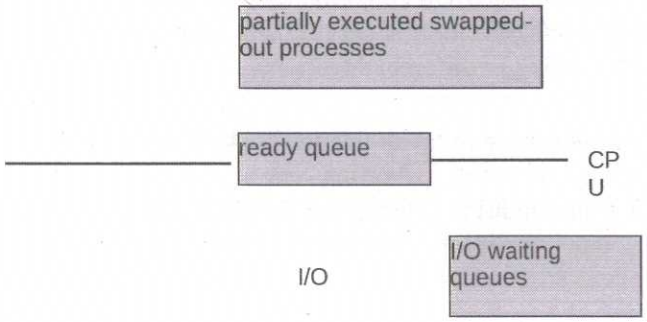
	 **Banker's Algorithm** Banker's Algorithm is resource allocation and deadlock avoidance algorithm which test all the request made by processes for resources. It check for safe state, after granting request, if the system remains in the safe state then it allows the request and if there is no safe state it don't allow the request made bythe process. • **Deadlock Detection** **1.If resources have single instance**: In this case for deadlock detection we can run an algorithm to check for cycle in the Resource Allocation Graph. Presence of cycle in the graph is the sufficient condition for deadlock in system with resources having single instance. **2. If there are multiple instances of resources**: Detection of cycle is necessary but not sufficient condition for deadlock detection with multiple resource instances , in this case system may or may not be in deadlock varies according to different situations. • **Deadlock recovery** Once a deadlock is detected, you will have to break the deadlock. It can be done through different ways, including, aborting one or more processes to break the circular wait condition causing the deadlock and preempting resources from one or more processes which are deadlocked. **Recovery methods** 1. Killing the process. Killing all the process involved in deadlock. Killing process one by one. After killing each process check for deadlock again keep repeating process till system recover from deadlock. disadvantage: lot of overhead; must re-run algorithm after each kill 2. Resource Preemption Resources are preempted from the processes involved in deadlock, preempted resources areallocated to other processes, so that there is a possibility of recovering the system from deadlock. In thiscase system go into starvation.	2 2		
VI(b)	**Schedulers** A process migrates among the various scheduling queues throughout its lifetime. The operating system must select, for scheduling purposes, processes from these queues in some fashion. The selection process is carried out by the appropriate scheduler. The **long-term scheduler**, or job scheduler, selects processes from this pool and loads them into memory for execution. The **short-term scheduler**, or CPU scheduler, selects from among the processes that are ready to execute and allocates the CPU to one of them.	1 1	7	



2

Some operating systems, such as time-sharing systems, may introduce an additional, intermediate level of scheduling. This **medium-term scheduler,** swaps out partially exected process and swaps in another process

1



2

VII(a) Reference String

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

9 **15**

<u>FIFO Page Replacement</u>

The simplest page-replacement algorithm is a first-in, first-out (FIFO) algorithm. A FIFO replacement algorithm associates with each page the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen.

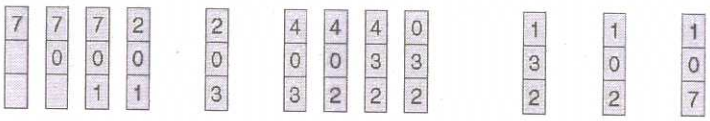
2

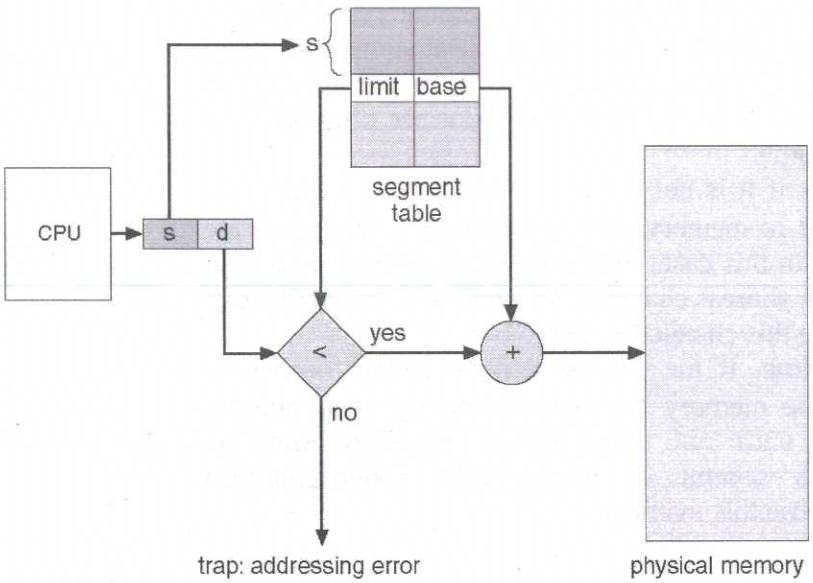
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

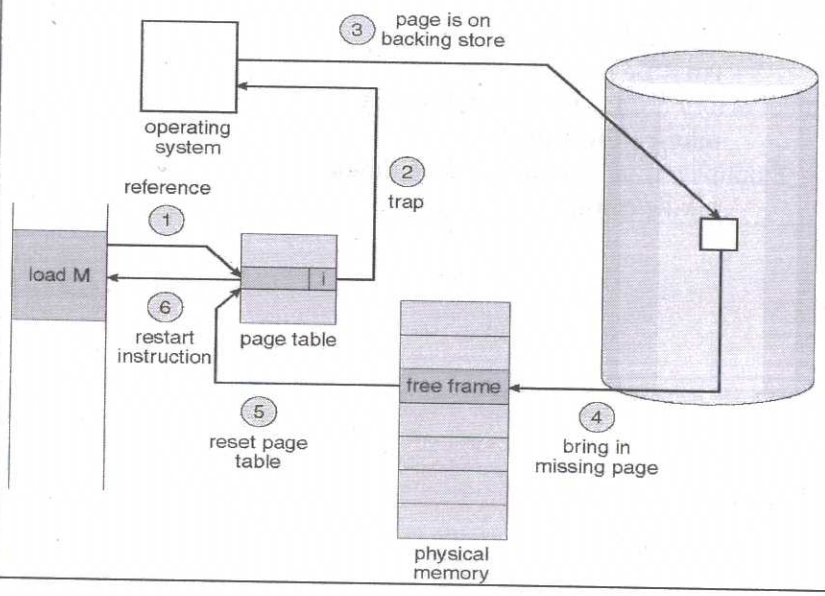
2.5

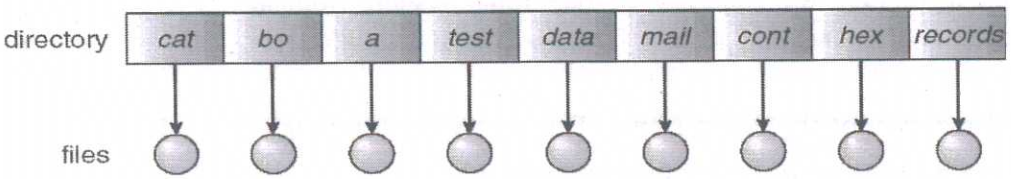
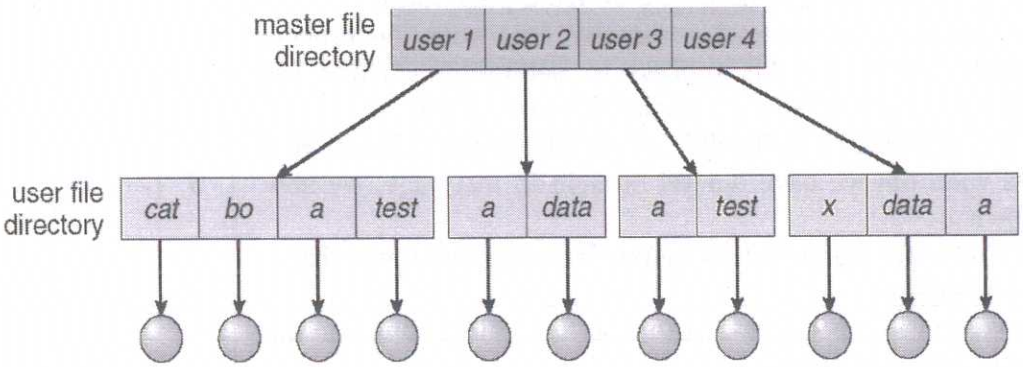
7	7	7	2	2	2	4	4	4	0	0	0	7	7	7
	0	0	0	3	3	3	2	2	2	1	1	1	0	0
		1	1	1	0	0	0	3	3	3	2	2	2	1

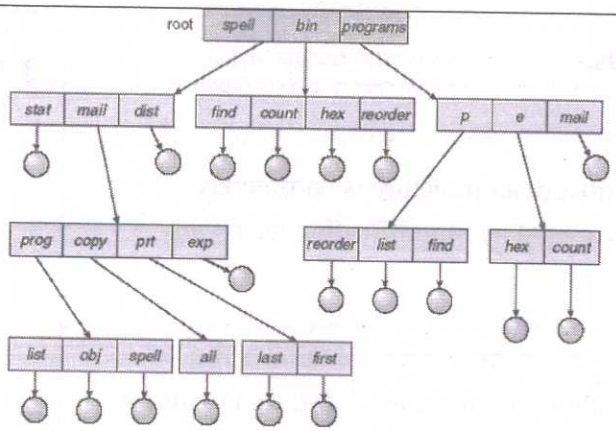
No of page faults : 15

	The FIFO page-replacement algorithm is easy to understand and program. However, its performance is not always good. LRU Page Replacement If we use the recent past as an approximation of the near future, then we can replace the page that ***has not been used*** for the longest period of time. This approach is the **least recently used (LRU) algorithm**. LRU replacement associates with each page the time of that page's last use. When a page must be replaced, LRU chooses the page that has not been used for the longest period of time. reference string 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1  page frames 12 faults – better than FIFO but worse than OPT Generally good algorithm and frequently used	2 2.5		
VII(b)	• **Compile time**. If we know at compile time where the process will reside in memory, then **absolute code** can be generated. For example, if we know that a user process will reside starting at location *R*, then the generated compiler code will start at that location and extend up from there. If, at some later time, the starting location changes, then it will be necessary to recompile this code. The MS-DOS .COM-format programs are bound at compile time. • **Load time**. If it is not known at compile time where the process will reside in memory, then the compiler must generate **relocatable code**. In this case, final binding is delayed until load time. If the starting address changes, we need only reload the user code to incorporate this changed value. • **Execution time**. If the process can be moved during its execution from one memory segment to another, then binding must be delayed until run time. Special hardware must be available for this scheme to work. Most general-purpose operating systems use this method.	1.5 1.5 1.5 1.5	6	

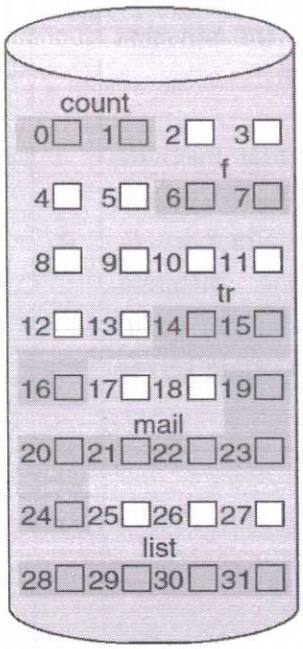
VIII(a)	**Segmentation** is a memory-management scheme that supports the programmer view of memory. A logical address space is a collection of segments. Segmentation Hardware  A logical address consists of a *two tuple:* <segment-number, offset>. ■ **Segment table** – maps two-dimensional physical addresses; each table entry has: **base** – contains the starting physical address where the segments reside in memory **limit** – specifies the length of the segment	1 4 1 1	**9**	**15**

	The segment number is used as an index to the segment table. The offset *d* of the logical address must be between 0 and the segment limit. If it is not, we trap to the operating system (logical addressing attempt beyond end of segment). When an offset is legal, it is added to the segment base to produce the address in physical memory of the desired byte. The segment table is thus essentially an array of base– limit register pairs.	2		
VIII(b)	**If there is a reference to a page, first reference to that page will trap to operating system. It is called a page fault.** 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 an invalid memory access. 2. If the reference was invalid, we terminate the process. If it was valid but we 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 newly allocated frame. 5. When the disk read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory. 6. We restart the instruction that was interrupted by the trap. The process can now access the page as though it had always been in memory.  	3 3	6	

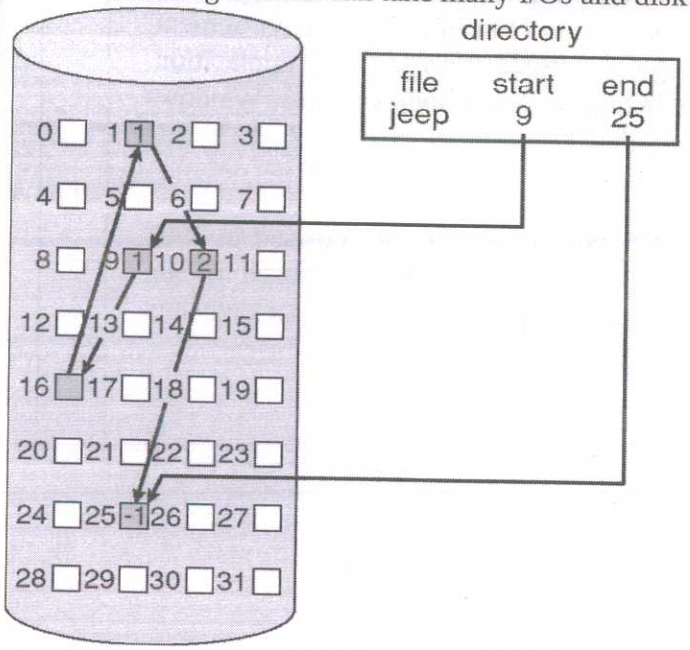
IX (a)	**1. Single level Directory** A single directory for all users  Naming problem Grouping problem **2. Two level Directory** Separate directory for each user  Path name Can have the same file name for different user Efficient searching No grouping capability **3. Tree structured Directories** **Absolute** or **relative** path name Creating a new file is done in current directory Delete a file **rm <file-name>** Creating a new subdirectory is done in current directory **mkdir <dir-name>** Example: if in current directory **/mail** **mkdir count**	3 3 3	9	15



IX (b)	***Hardware virtualization***		6	
	The most common type of virtualization today, hardware virtualization is made possible by a virtual machine manager (VM) called the "hypervisor". The hypervisor creates virtual versions of computers and operating systems and consolidates them into one large physical server, so that all the hardware resources can be utilized more efficiently. It also enables users to run different operating systems on the same machine simultaneously.			
	It involves embedding virtual machine software into the server's hardware components. That software is called the hypervisor. The hypervisor manages the shared physical hardware resources between the guest OS & the host OS. The abstracted hardware is represented as actual hardware. Virtualization means abstraction & hardware virtualization is achieved by abstracting the physical hardware part using Virtual Machine Monitor (VMM) or hypervisor.			
	The term hardware virtualization is used when VMM or virtual machine software or any hypervisor gets directly installed on the hardware system. The primary task of the hypervisor is to process monitoring, memory & hardware controlling. After hardware virtualization is done, different operating systems can be installed, and various applications can run on it. Hardware virtualization, when done for server platforms, is also called server virtualization.	3		
	Advantages of hardware virtualization • Lower Cost: • Efficient resource utilization: • Increase IT flexibility: • Advanced Hardware Virtualization features:			
	Types of hardware virtualization			

	Hardware virtualization is of three kinds. These are: • Full Virtualization: Here the hardware architecture is completely simulated. Guest software doesn't need any modification to run any applications. • Emulation Virtualization: Here the virtual machine simulates the hardware & is independent. Furthermore, the guest OS doesn't require any modification. • Para-Virtualization: Here, the hardware is not simulated; instead the guest software runs its isolated system.	3		
X (a)	File Allocation methods 1. Contiguous 2. Linked 3. Indexed **Contiguous allocation** requires that each file occupy a set of contiguous blocks on the disk. Disk addresses define a linear ordering on the disk.Contiguous allocation of a file is defined by the disk address and length (in block units) of the first block. Accessing a file that has been allocated contiguously is easy  directory file start length count 0 2 tr 14 3 mail 19 6 list 28 4 f 6 2 **Linked allocation** – each file a linked list of blocks • File ends at nil pointer • No external fragmentation • Each block contains pointer to next block • No compaction, external fragmentation • Free space management system called when new block needed • Improve efficiency by clustering blocks into groups but increases internal fragmentation	3	9	15

- Reliability can be a problem
- Locating a block can take many I/Os and disk seeks



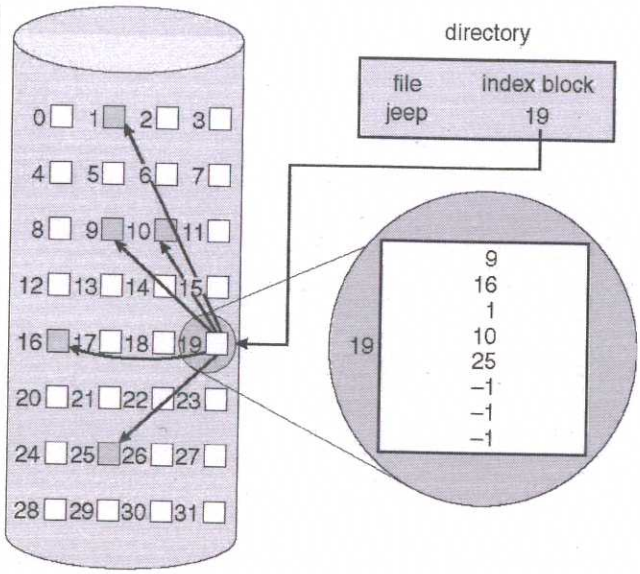
Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file.

3

Indexed Allocation

Linked allocation solves the external-fragmentation and size-declaration problems of contiguous allocation. However, in the absence of a FAT, linked allocation cannot support efficient direct access, since the pointers to the blocks are scattered with the blocks themselves all over the disk and must be retrieved in order. **Indexed allocation** solves this problem by bringing all the pointers together into one location: the **index block**.

Each file has its own index block, which is an array of disk-block addresses. The *ith* entry in the index block points to the *ith* block of the file. The directory contains the address of the index block. To find and read the *ith* block, we use the pointer in the *ith* index-block entry.



3

X (b)	A **VirtualBox** or **VB** is a software virtualization package that installs on an operating system as an application. VirtualBox allows additional operating systems to be installed on it, as a Guest OS, and run in a virtual environment. In 2010, VirtualBox was the most popular virtualization software application. Supported operating systems include Windows XP, Windows Vista, Windows 7, macOS X, Linux, Solaris, and OpenSolaris. VirtualBox was originally developed by Innotek GmbH and released in 2007 as an open-source software package. The company was later purchased by Sun Microsystems. Oracle Corporation now develops the software package and titles it Oracle VM VirtualBox.	**6**	**6**	