

Sahrdaya College of Engineering and Technology, Kodakara																										
Answer Key																										
APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY																										
FOURTH SEMESTER B.TECH DEGREE EXAMINATION, JULY 2021 (2019 Scheme)																										
Course Code: <b>CST206</b>																										
Course Name: <b>OPERATING SYSTEMS</b>																										
PART A																										
	<b>Answer all questions. Each question carries 3 marks</b>																									
Q1	Which are the three methods used to pass parameters to operating system ?																									
Ans	There are three main methods to pass the parameters required for a system call: (1) Pass the parameters in registers (this may prove insufficient when there are more parameters than registers). (2) Store the parameters in a block, or table, in memory, and pass the address of block as a parameter in a register. This approach is used by Linux and Solaris. (3) Push the parameters onto a stack; to be popped off by the OS. Block and stack methods do not limit the number or length of parameters passed.																									
Q2	Write three advantages of peer-to-peer system over client server system.																									
Ans	<ul style="list-style-type: none"> <li>• No need for a network administrator.</li> <li>• Network is fast and inexpensive to setup and maintain.</li> <li>• Each computer can make backup copies of its data to other computers for security.</li> <li>• Peer-to-peer is, by far the easiest type of network to build for either home or office use.</li> </ul>																									
Q3	Differentiate Pre-emptive and Non-pre-emptive scheduling giving the application of each of them																									
Ans	<table border="1"> <thead> <tr> <th>Parameter</th><th>PREEMPTIVE SCHEDULING</th><th>NON-PREEMPTIVE SCHEDULING</th></tr> </thead> <tbody> <tr> <td>Basic</td><td>In this resources(CPU Cycle) are allocated to a process for a limited time.</td><td>Once resources(CPU Cycle) are allocated to a process, the process holds it till its burst time or switches to waiting state.</td></tr> <tr> <td>Interrupt</td><td>Process can be interrupted in between.</td><td>Process can not be interrupted until its time is up.</td></tr> <tr> <td>Starvation</td><td>If a process having high priority frequently arrives in the ready queue, a low priority process may starve.</td><td>If a process with a long burst time arrives first, then later coming process may starve.</td></tr> <tr> <td>Overhead</td><td>It has overheads of scheduling the processes.</td><td>It does not have overheads.</td></tr> <tr> <td>Flexibility</td><td>flexible</td><td>rigid</td></tr> <tr> <td>Cost</td><td>cost associated</td><td>no cost associated</td></tr> <tr> <td>CPU Utilization</td><td>In preemptive scheduling, CPU utilization is high.</td><td>It is low in non preemptive scheduling.</td></tr> </tbody> </table>		Parameter	PREEMPTIVE SCHEDULING	NON-PREEMPTIVE SCHEDULING	Basic	In this resources(CPU Cycle) are allocated to a process for a limited time.	Once resources(CPU Cycle) are allocated to a process, the process holds it till its burst time or switches to waiting state.	Interrupt	Process can be interrupted in between.	Process can not be interrupted until its time is up.	Starvation	If a process having high priority frequently arrives in the ready queue, a low priority process may starve.	If a process with a long burst time arrives first, then later coming process may starve.	Overhead	It has overheads of scheduling the processes.	It does not have overheads.	Flexibility	flexible	rigid	Cost	cost associated	no cost associated	CPU Utilization	In preemptive scheduling, CPU utilization is high.	It is low in non preemptive scheduling.
Parameter	PREEMPTIVE SCHEDULING	NON-PREEMPTIVE SCHEDULING																								
Basic	In this resources(CPU Cycle) are allocated to a process for a limited time.	Once resources(CPU Cycle) are allocated to a process, the process holds it till its burst time or switches to waiting state.																								
Interrupt	Process can be interrupted in between.	Process can not be interrupted until its time is up.																								
Starvation	If a process having high priority frequently arrives in the ready queue, a low priority process may starve.	If a process with a long burst time arrives first, then later coming process may starve.																								
Overhead	It has overheads of scheduling the processes.	It does not have overheads.																								
Flexibility	flexible	rigid																								
Cost	cost associated	no cost associated																								
CPU Utilization	In preemptive scheduling, CPU utilization is high.	It is low in non preemptive scheduling.																								

		Examples	Examples of preemptive scheduling are Round Robin and Shortest Remaining Time First.	Examples of non-preemptive scheduling are First Come First Serve and Shortest Time First.
	Q4	Why is context switching considered to be an overhead to the system?		
	Ans	Context Switching leads to an overhead cost because of TLB flushes, sharing the cache between multiple tasks, running the task scheduler etc. Context switching between two threads of the same process is faster than between two different processes as threads have the same virtual memory maps. Because of this TLB flushing is not required.		
	Q5	What are necessary conditions which can lead to a deadlock situation in a system?		
	Ans	<p>Four conditions that must hold for a deadlock to be possible:</p> <ul style="list-style-type: none"> <li>• Mutual exclusion: processes require exclusive control of its resources (not sharing).</li> <li>• Hold and wait: process may wait for a resource while holding others.</li> <li>• No preemption: process will not give up a resource until it is finished with it. Also, processes are irreversible: unable to reset to an earlier state where resources not held.</li> <li>• Circular wait: each process in the chain holds a resource requested by another</li> </ul>		
	Q6	Explain the wait and signal operations used in semaphores.		
	Ans	<p>Two standard operations, wait and signal are defined on the semaphore. Entry to the critical section is controlled by the wait operation and exit from a critical region is taken care by signal operation. The wait, signal operations are also called P and V operations. The manipulation of semaphore (S) takes place as following:</p> <ol style="list-style-type: none"> <li>1. The wait command P(S) decrements the semaphore value by 1. If the resulting value becomes negative then P command is delayed until the condition is satisfied.</li> <li>2. The V(S) i.e. signals operation increments the semaphore value by 1.</li> </ol> <p>Mutual exclusion on the semaphore is enforced within P(S) and V(S). If a number of processes attempt P(S) simultaneously, only one process will be allowed to proceed &amp; the other processes will be waiting. These operations are defined as under –</p> <div style="border: 1px solid black; padding: 10px; margin-top: 10px;"> <p>P(S) or wait(S):</p> <p>If <math>S &gt; 0</math> then</p> <p style="padding-left: 20px;">Set S to S-1</p> <p>Else</p> <p style="padding-left: 20px;">Block the calling process (i.e. Wait on S)</p> <p>V(S) or signal(S):</p> <p>If any processes are waiting on S</p> <p style="padding-left: 20px;">Start one of these processes</p> <p>Else</p> <p style="padding-left: 20px;">Set S to S+1</p> </div>		

		<p>The semaphore operation are implemented as operating system services and so wait and signal are atomic in nature i.e. once started, execution of these operations cannot be interrupted.</p>
	Q7	How does swapping result in better memory management?
	Ans	<p>Memory swapping works by making use of <u>virtual memory</u> and storage space in an approach that provides additional resources when required. In short, this additional memory enables the computer to run faster and crunch data better.</p> <p>With memory swapping, the operating system makes use of storage disk space to provide the functional equivalent of memory storage execution space. The space on the storage device is referred to as “swap space” and is used to run processes that have been swapped out of main physical memory.</p> <p>The process of memory swapping is managed by an operating system or by a virtual machine hypervisor. Swapping is often enabled by default, though users can choose to disable the capability.</p> <p>The actual memory swapping process and the creation of a swap file is automatically managed by the operating system. It is initiated when needed as physical RAM is used and additional capacity is required by processes and applications. As additional RAM is required, the state of the physical memory page is mapped to the swap space, enabling a form of virtual (non-physical RAM) memory capacity.</p> <p>In other words, the main purpose of swapping in memory management is to enable more usable memory than held by the computer hardware.</p> <p>There are times when physical memory will be allocated and a process needs additional memory. Rather than limiting a system to only having memory that is based on physical RAM, memory swapping enables operating systems and their users to extend memory to disk.</p> <p>Memory swapping is an essential component of modern memory management helping to ensure availability and overall system stability.</p>
	Q8	Explain the concept of virtual memory. Write one memory management scheme which supports virtual memory
	Ans	<p>Virtual memory is a memory management technique where secondary memory can be used as if it were a part of the main memory. Virtual memory is a common technique used in a computer's operating system (OS).</p>

Virtual memory uses both hardware and software to enable a computer to compensate for physical memory shortages, temporarily transferring data from random access memory (RAM) to disk storage. Mapping chunks of memory to disk files enables a computer to treat secondary memory as though it were main memory.

Today, most personal computers (PCs) come with at least 8 GB (gigabytes) of RAM. But, sometimes, this is not enough to run several programs at one time. This is where virtual memory comes in. Virtual memory frees up RAM by swapping data that has not been used recently over to a storage device, such as a hard drive or solid-state drive (SSD).

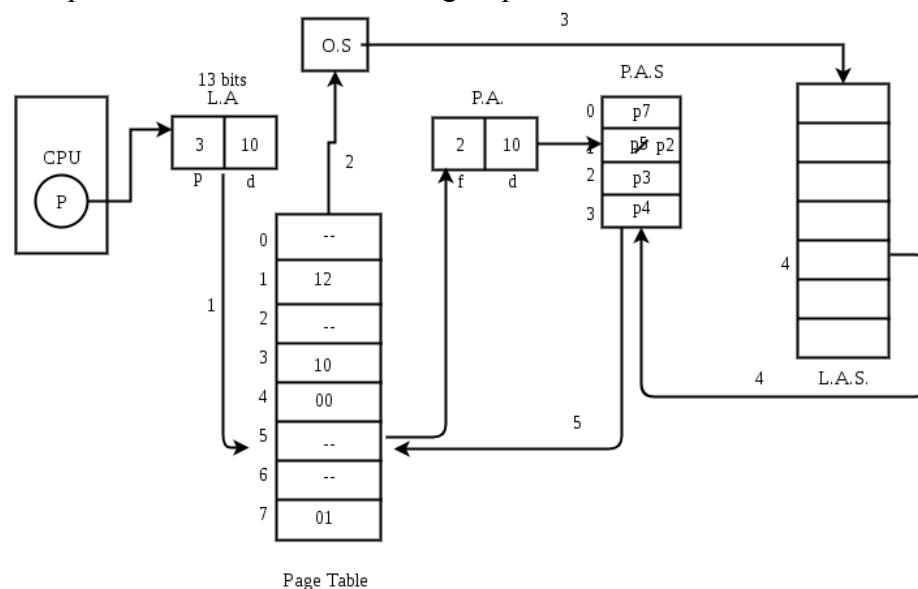
Virtual memory is important for improving system performance, multitasking and using large programs. However, users should not overly rely on virtual memory, since it is considerably slower than RAM. If the OS has to swap data between virtual memory and RAM too often, the computer will begin to slow down -- this is called thrashing.

Virtual memory was developed at a time when physical memory -- also referenced as RAM -- was expensive. Computers have a finite amount of RAM, so memory will eventually run out when multiple programs run at the same time. A system using virtual memory uses a section of the hard drive to emulate RAM. With virtual memory, a system can load larger or multiple programs running at the same time, enabling each one to operate as if it has more space, without having to purchase more RAM.

### **Demand Paging :**

The process of loading the page into memory on demand (whenever page fault occurs) is known as demand paging.

The process includes the following steps :



1. If the CPU tries to refer to a page that is currently not available in the main memory, it generates an interrupt indicating a memory access fault.
2. The OS puts the interrupted process in a blocking state. For the execution to proceed the OS must bring the required page into the memory.
3. The OS will search for the required page in the logical address space.

		<p>4. The required page will be brought from logical address space to physical address space. The page replacement algorithms are used for the decision-making of replacing the page in physical address space.</p> <p>5. The page table will be updated accordingly.</p> <p>6. The signal will be sent to the CPU to continue the program execution and it will place the process back into the ready state.</p> <p>Hence whenever a page fault occurs these steps are followed by the operating system and the required page is brought into memory.</p> <p><b>Advantages :</b></p> <ul style="list-style-type: none"> <li>• More processes may be maintained in the main memory: Because we are going to load only some of the pages of any particular process, there is room for more processes. This leads to more efficient utilization of the processor because it is more likely that at least one of the more numerous processes will be in the ready state at any particular time.</li> <li>• A process may be larger than all of the main memory: One of the most fundamental restrictions in programming is lifted. A process larger than the main memory can be executed because of demand paging. The OS itself loads pages of a process in the main memory as required.</li> <li>• It allows greater multiprogramming levels by using less of the available (primary) memory for each process.</li> </ul>
	Q9	Compare sequential access and direct access methods of storage devices.
	Ans	<p>On the basis of data is stored in the memory and there access approach . There are four type of access Methods</p> <ol style="list-style-type: none"> <li>1. Sequential Access Methods</li> <li>2. Direct Access Methods</li> <li>3. Random Access Methods</li> <li>4. Associative Access Methods</li> </ol> <p><b>1. Sequential Access Method:</b></p> <p>Memory is organised into units of records, called record and each record is associated with address. In sequential access method access must be made in linear sequence. A shared read write mechanism is used and for access we traverse through all locations starting from the current location to desired location , passing and rejecting all the intermediate records until target record is not found. Magnetic tape is the example where sequential access method is used. In Simple Data Structure Linked List is the example of Sequential access.</p> <p><b>2. Direct Access:</b></p> <p>In direct access data is read immediately without iterating from the beginning. With the sequential access, direct access involves shared read write mechanism and a unique address is assigned to each data or records. Access is made by using this address and data</p>

		or record is accessed immediately without sequentially searching the complete block of memory. Disk are of Direct Access.
	Q10	Write notes on disk formatting
	Ans	<p><b>Disk Formatting</b></p> <p>Before a disk can be used, it has to be low-level formatted, which means laying down all of the headers and trailers marking the beginning and ends of each sector. Included in the header and trailer are the linear sector numbers, and error-correcting codes, ECC, which allow damaged sectors to not only be detected, but in many cases for the damaged data to be recovered ( depending on the extent of the damage. ) Sector sizes are traditionally 512 bytes, but may be larger, particularly in larger drives.</p> <p>ECC calculation is performed with every disk read or write, and if damage is detected but the data is recoverable, then a soft error has occurred. Soft errors are generally handled by the on-board disk controller, and never seen by the OS.</p> <p>Once the disk is low-level formatted, the next step is to partition the drive into one or more separate partitions. This step must be completed even if the disk is to be used as a single large partition, so that the partition table can be written to the beginning of the disk. After partitioning, then the filesystems must be logically formatted, which involves laying down the master directory information ( FAT table or inode structure ), initializing free lists, and creating at least the root directory of the filesystem. ( Disk partitions which are to be used as raw devices are not logically formatted. This saves the overhead and disk space of the filesystem structure, but requires that the application program manage its own disk storage requirements. )</p>
<b>PART B</b>		
<i>Answer any one full question from each module. Each question carries 14 marks</i>		
<b>Module 1</b>		
	Q11	<p>a) Distinguish among the following terminologies associated with the operating system and explain each of them in detail. (i) Multiprogramming systems (ii) Multitasking systems (iii) Multiprocessor systems.</p> <p>b) Explain, how the long-term scheduler directly affects the system performance</p>
	Ans	<p>a) Vvv</p> <p><b>Multiprogramming</b> – A computer running more than one program at a time (like running Excel and Firefox simultaneously).</p> <p><b>Multiprocessing</b> – A computer using more than one CPU at a time.</p>

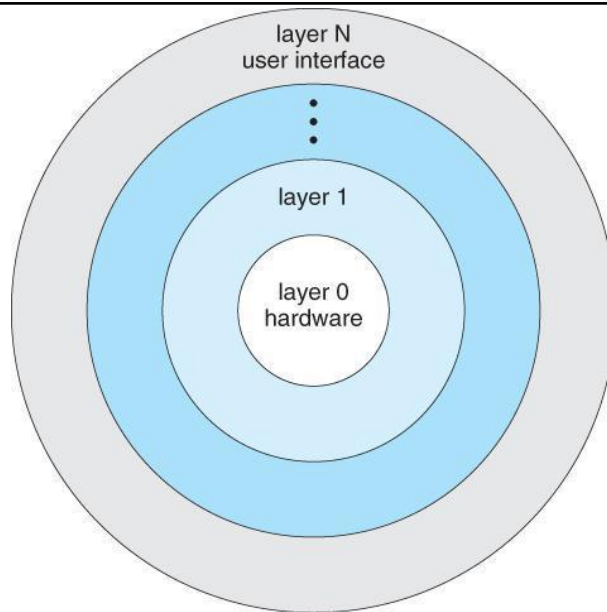
	<p><b>Multitasking</b> – Tasks sharing a common resource (like 1 CPU).</p> <p><b>(i) Multi programming –</b></p> <p>In a modern computing system, there are usually several concurrent application processes which want to execute. Now it is the responsibility of the Operating System to manage all the processes effectively and efficiently.</p> <p>One of the most important aspects of an Operating System is to multi program.</p> <p>In a computer system, there are multiple processes waiting to be executed, i.e. they are waiting when the CPU will be allocated to them and they begin their execution. These processes are also known as jobs. Now the main memory is too small to accommodate all of these processes or jobs into it. Thus, these processes are initially kept in an area called job pool. This job pool consists of all those processes awaiting allocation of main memory and CPU.</p> <p>CPU selects one job out of all these waiting jobs, brings it from the job pool to main memory and starts executing it. The processor executes one job until it is interrupted by some external factor or it goes for an I/O task.</p> <p><b>(ii) Multitasking</b></p> <p>As the name itself suggests, multi tasking refers to execution of multiple tasks (say processes, programs, threads etc.) at a time. In the modern operating systems, we are able to play MP3 music, edit documents in Microsoft Word, surf the Google Chrome all simultaneously, this is accomplished by means of multi tasking.</p> <p>Multitasking is a logical extension of multi programming. The major way in which multitasking differs from multi programming is that multi programming works solely on the concept of context switching whereas multitasking is based on time sharing alongside the concept of context switching.</p> <p>Multi tasking system's working –</p> <p>In a time sharing system, each process is assigned some specific quantum of time for which a process is meant to execute. Say there are 4 processes P1, P2, P3, P4 ready to execute. So each of them are assigned some time quantum for which they will execute e.g time quantum of 5 nanoseconds (5 ns). As one process begins execution (say P2), it executes for that quantum of time (5 ns). After 5 ns the CPU starts the execution of the other process (say P3) for the specified quantum of time.</p> <p>Thus the CPU makes the processes to share time slices between them and execute accordingly. As soon as time quantum of one process expires, another process begins its execution.</p> <p>Here also basically a context switch is occurring but it is occurring so fast that the user is able to interact with each program separately while it is running.</p> <p>This way, the user is given the illusion that multiple processes/ tasks are executing simultaneously. But actually only one process/ task is executing at a</p>
--	--

	<p>particular instant of time. In multitasking, time sharing is best manifested because each running process takes only a fair quantum of the CPU time.</p> <p>(iii) <b>Multiprocessing:</b>          In a uni-processor system, only one process executes at a time. Multiprocessing is the use of two or more CPUs (processors) within a single Computer system. The term also refers to the ability of a system to support more than one processor within a single computer system. Now since there are multiple processors available, multiple processes can be executed at a time. These multi processors share the computer bus, sometimes the clock, memory and peripheral devices also.</p> <p>Multi processing system's working –</p> <p>With the help of multiprocessing, many processes can be executed simultaneously. Say processes P1, P2, P3 and P4 are waiting for execution. Now in a single processor system, firstly one process will execute, then the other, then the other and so on.</p> <p>But with multiprocessing, each process can be assigned to a different processor for its execution. If its a dual-core processor (2 processors), two processes can be executed simultaneously and thus will be two times faster, similarly a quad core processor will be four times as fast as a single processor.</p> <p>b) Long-Term schedulers are those schedulers whose decision will have a long-term effect on the performance. The duty of the long-term scheduler is to bring the process from the JOB pool to the Ready state for its execution.</p> <p>Long-Term Scheduler is also called Job Scheduler and is responsible for controlling the Degree of Multiprogramming i.e. the total number of processes that are present in the ready state.</p> <p>So, the long-term scheduler decides which process is to be created to put into the ready state.</p> <p><b><u>Effect on performance</u></b></p> <p>i. The long term scheduler is responsible for creating a balance between the I/O bound(a process is said to be I/O bound if the majority of the time is spent on the I/O operation) and CPU bound(a process is said to be CPU bound if the majority of the time is spent on the CPU). So, if we create processes which are all I/O bound then the CPU might not be used and it will remain idle for most of the time. This is because the majority of the time will be spent on the I/O operation.</p> <p>ii. So, if we create processes that are having high a CPU bound or a perfect balance between the I/O and CPU bound, then the overall performance of the system will be increased.</p>
--	---



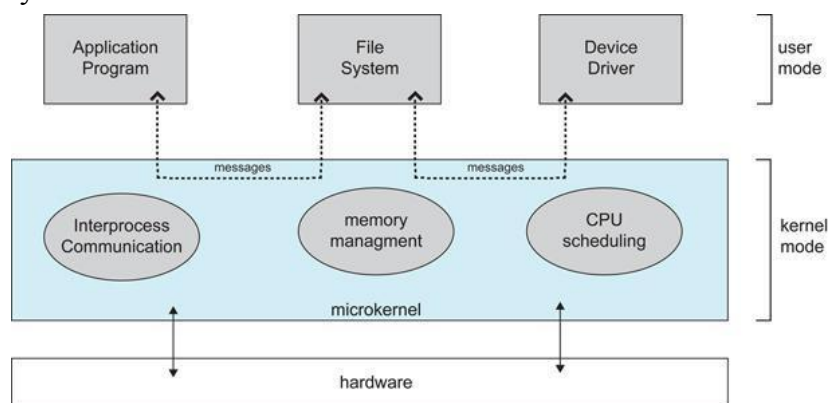
Q12	<p>a) Explain in detail about the various functions of operating systems</p> <p>b) Write notes on the following operating system structures. (i) Layered approach (ii) Microkernel</p>
Ans	<p>a) <b>Important functions of an operating System:</b></p> <ol style="list-style-type: none"> <li>1. <b>Security –</b> The operating system uses password protection to protect user data and similar other techniques. it also prevents unauthorized access to programs and user data.</li> <li>2. <b>Control over system performance –</b> Monitors overall system health to help improve performance. records the response time between service requests and system response to having a complete view of the system health. This can help improve performance by providing important information needed to troubleshoot problems.</li> <li>3. <b>Job accounting –</b> Operating system Keeps track of time and resources used by various tasks and users, this information can be used to track resource usage for a particular user or group of users.</li> <li>4. <b>Error detecting aids –</b> The operating system constantly monitors the system to detect errors and avoid the malfunctioning of a computer system.</li> <li>5. <b>Coordination between other software and users –</b> Operating systems also coordinate and assign interpreters, compilers, assemblers, and other software to the various users of the computer systems.</li> <li>6. <b>Memory Management –</b> The operating system manages the Primary Memory or Main Memory. Main memory is made up of a large array of bytes or words where each byte or word is assigned a certain address. Main memory is fast storage and it can be accessed directly by the CPU. For a program to be executed, it should be first loaded in the main memory. An Operating System performs the following activities for memory management: It keeps track of primary memory, i.e., which bytes of memory are used by which user program. The memory addresses that have already been allocated and the memory addresses of the memory that has not yet been used. In multiprogramming, the OS decides the order in which processes are granted access to memory, and for how long. It Allocates the memory to a process when the process requests it and deallocates the memory when the process has terminated or is performing an I/O operation.</li> <li>7. <b>Processor Management –</b> In a multi-programming environment, the OS decides the order in which processes have access to the processor, and how much processing time each process has. This function of OS is called process scheduling. An</li> </ol>

	<p>Operating System performs the following activities for processor management.</p> <p>Keeps track of the status of processes. The program which performs this task is known as a traffic controller. Allocates the CPU that is a processor to a process. De-allocates processor when a process is no more required.</p> <p>8.     <b>Device Management –</b>  An OS manages device communication via their respective drivers. It performs the following activities for device management. Keeps track of all devices connected to the system. designates a program responsible for every device known as the Input/Output controller. Decides which process gets access to a certain device and for how long. Allocates devices in an effective and efficient way. Deallocates devices when they are no longer required.</p> <p>9.     <b>File Management –</b>  A file system is organized into directories for efficient or easy navigation and usage. These directories may contain other directories and other files. An Operating System carries out the following file management activities. It keeps track of where information is stored, user access settings and status of every file, and more... These facilities are collectively known as the file system.</p> <p>b) ..</p> <p>(i)    <b>Layered Approach</b></p> <ul style="list-style-type: none"> <li>● Another approach is to break the OS into a number of smaller layers, each of which rests on the layer below it, and relies solely on the services provided by the next lower layer.</li> <li>● This approach allows each layer to be developed and debugged independently, with the assumption that all lower layers have already been debugged and are trusted to deliver proper services.</li> <li>● The problem is deciding what order in which to place the layers, as no layer can call upon the services of any higher layer, and so many chicken-and-egg situations may arise.</li> <li>● Layered approaches can also be less efficient, as a request for service from a higher layer has to filter through all lower layers before it reaches the HW, possibly with significant processing at each step.</li> </ul>
--	---



(ii) **Microkernel**

- The basic idea behind micro kernels is to remove all non-essential services from the kernel, and implement them as system applications instead, thereby making the kernel as small and efficient as possible.
- Most microkernels provide basic process and memory management, and message passing between other services, and not much more.
- Security and protection can be enhanced, as most services are performed in user mode, not kernel mode.
- System expansion can also be easier, because it only involves adding more system applications, not rebuilding a new kernel.
- Mach was the first and most widely known microkernel, and now forms a major component of Mac OSX.
- Windows NT was originally microkernel, but suffered from performance problems relative to Windows 95. NT 4.0 improved performance by moving more services into the kernel, and now XP is back to being more monolithic.
- Another microkernel example is QNX, a real-time OS for embedded systems.



## Module 2

Q13	<p>a) A writer process like to send some bulk information to a reader process. Explain the IPC mechanism that can be used for this purpose.</p> <p>b) How many child process will be created for the following code ?  <pre>void main() { fork(); fork(); printf("HELLO\n"); fork(); printf("WELCOME\n"); }</pre> How many times HELLO and WELCOME will be printed? Justify your answer.</p>
Ans	<p>a) The communication between these processes can be seen as a method of co-operation between them. Processes can communicate with each other through both: Shared Memory &amp; Message Passing.  Here, in this context message passing mechanism can be used.</p> <p>In this method, processes communicate with each other without using any kind of shared memory. If two processes p1 and p2 want to communicate with each other, they proceed as follows:</p> <ul style="list-style-type: none"> <li>Establish a communication link (if a link already exists, no need to establish it again.)</li> <li>Start exchanging messages using basic primitives.</li> </ul> <p>We need at least two primitives:</p> <ul style="list-style-type: none"> <li><b>send</b>(message, destination) or <b>send</b>(message)</li> <li><b>receive</b>(message, host) or <b>receive</b>(message)</li> </ul> <div data-bbox="319 1120 1181 1702" data-label="Diagram"> <pre> graph TD     SP[Sending Process] --&gt; MP1[Message passing module]     MP1 --&gt; RP[Receiving Process]     RP --&gt; MP2[Message passing module]     MP2 --&gt; SP     </pre> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> <span style="border-right: 1px solid black; padding-right: 10px;">ProcessId</span> <span>Message</span> </div> </div> <p>The message size can be of fixed size or of variable size. If it is of fixed size, it is easy for an OS designer but complicated for a programmer and if it is of variable size then it is easy for a programmer but complicated for the OS designer. A standard message can have two parts: <b>header and body</b>.  The <b>header part</b> is used for storing message type, destination id, source id, message length, and control information. The control information contains information like what to do if runs out of buffer space, sequence number, priority. Generally, message is sent using FIFO style.</p>

		<p>b)</p> <ul style="list-style-type: none"> <li>• 7 child process will be created</li> <li>• HELLO will be printed 4 times</li> <li>• WELCOME will be printed 8 times</li> </ul>																		
Q14		<p>a) Five batch jobs A through E arrive at a computer system in the order A to E at almost the same time. They have estimated running times of 6, 4, 1, 3, and 7 seconds. Their (externally determined) priorities are 3, 5, 2, 1, and 4 respectively, with 5 being the highest priority. For each of the following scheduling algorithms, determine the average process waiting time. Ignore process switching overhead.</p> <p>(i) Round Robin (assume quantum = 2 sec) (ii) First-come first-served (iii) Shortest job first (iv) Priority scheduling</p> <p>b) Point out the significance of Zero capacity queue in IPC?</p>																		
Ans		<p>a) The burst times and priorities of the processes are:</p> <table border="1"> <thead> <tr> <th>Process</th><th>Burst Times</th><th>Priorities</th></tr> </thead> <tbody> <tr> <td>A</td><td>6</td><td>3</td></tr> <tr> <td>B</td><td>4</td><td>5</td></tr> <tr> <td>C</td><td>1</td><td>2</td></tr> <tr> <td>D</td><td>3</td><td>1</td></tr> <tr> <td>E</td><td>7</td><td>4</td></tr> </tbody> </table> <p>(i) RR with quantum = 2</p> <p>Gantt Chart:</p> <p>So the average waiting time is given by: RR waiting time=10</p> <p>(ii) FCFS:</p> <p>The processes are scheduled in the order: A B C D E</p> $  \begin{aligned}  T_{wait} &= (0 + 6 + (6+4) + (6+4+1) + (6+4+1+3)) / 5 \\  &= (0 + 6 + 10 + 11 + 14) / 5 \\  &= 41 / 5 \\  &= 8.2 \text{ ( minutes )}  \end{aligned}  $ <p>(iii) SJF:</p>	Process	Burst Times	Priorities	A	6	3	B	4	5	C	1	2	D	3	1	E	7	4
Process	Burst Times	Priorities																		
A	6	3																		
B	4	5																		
C	1	2																		
D	3	1																		
E	7	4																		

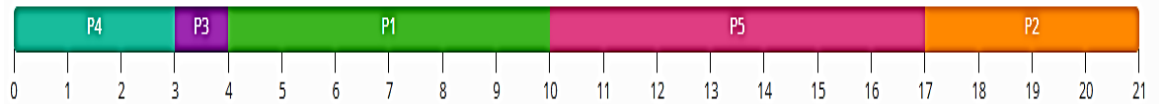
The processes are scheduled in the order: C D B A E

$$T_{wait} = ((1+4+3) + (1+3) + 0 + (1) + (1+4+3+6)) / 5 = (8 + 4 + 0 + 1 + 14) / 5$$

$$= 27 / 5$$

$$= 5.4 \text{ ( minutes )}$$

(iv) Priority:



The processes are scheduled in the order: B E A C D

Similarly, the waiting time for A B C D E are: 11, 0, 17, 18, 4 respectively.

So the average waiting time is;

$T_{wait}$	$= (11 + 0 + 17 + 18 + 4)$
	$/5$
	$= 50 / 5$
	$= \mathbf{10 \text{ ( minutes )}}$

b) In this case, the sender must block until the recipient receives the message.

### Module 3

Q15

a) Consider the following snapshot of a system with five processes P1, P2, P3, P4, P5 and four resources A, B, C, D. What is the total number of instances of A, B, C, and D? Using Bankers Algorithm check whether the system is in safe state or not.

	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P1	1	0	2	2	3	2	5	2	3	0	0	1
P2	0	2	1	2	3	4	1	2				
P3	2	4	5	0	2	7	7	3				
P4	3	0	0	0	5	5	0	7				
P5	4	2	1	3	6	2	1	4				

b) What is critical section problem? What are the requirements that need to be satisfied by any solution to critical section problem? Give a solution to a 2 process critical section problem.

Ans

a)  
Total instances (A, B, C, D) = (13, 8, 9, 8)  
Need matrix:

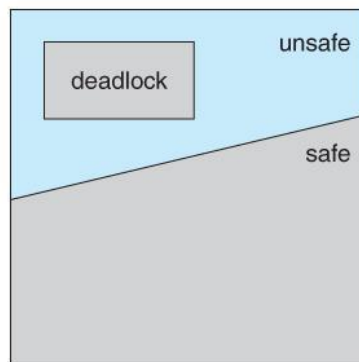
	A	B	C	D
--	---	---	---	---

P1	2	2	3	0
P2	3	2	0	0
P3	0	3	2	3
P4	2	5	0	7
P5	2	0	0	1

The system is in safe state

### **Safe State**

- A state is safe if the system can allocate all resources requested by all processes ( up to their stated maximums ) without entering a deadlock state.
- More formally, a state is safe if there exists a safe sequence of processes { P0, P1, P2, ..., PN } such that all of the resource requests for Pi can be granted using the resources currently allocated to Pi and all processes Pj where j < i. ( I.e. if all the processes prior to Pi finish and free up their resources, then Pi will be able to finish also, using the resources that they have freed up. )
- If a safe sequence does not exist, then the system is in an unsafe state, which MAY lead to deadlock. ( All safe states are deadlock free, but not all unsafe states lead to deadlocks. )



### **Safety Algorithm**

In order to apply the Banker's algorithm, we first need an algorithm for determining whether or not a particular state is safe.

This algorithm determines if the current state of a system is safe, according to the following steps:

1. Let Work and Finish be vectors of length m and n respectively.
  - Work is a working copy of the available resources, which will be modified during the analysis.
  - Finish is a vector of booleans indicating whether a particular process can finish. ( or has finished so far in the analysis. )
  - Initialize Work to Available, and Finish to false for all elements.

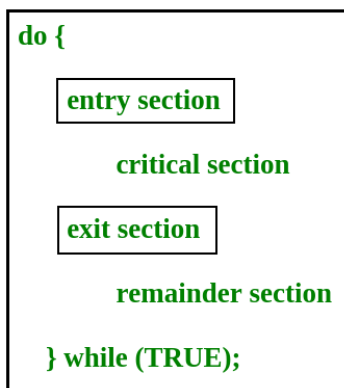
2. Find an  $i$  such that both (A)  $\text{Finish}[i] == \text{false}$ , and (B)  $\text{Need}[i] < \text{Work}$ . This process has not finished, but could with the given available working set. If no such  $i$  exists, go to step 4.
3. Set  $\text{Work} = \text{Work} + \text{Allocation}[i]$ , and set  $\text{Finish}[i]$  to true. This corresponds to process  $i$  finishing up and releasing its resources back into the work pool. Then loop back to step 2.
4. If  $\text{finish}[i] == \text{true}$  for all  $i$ , then the state is a safe state, because a safe sequence has been found.

- b) Critical Section is the part of a program which tries to access shared resources. That resource may be any resource in a computer like a memory location, Data structure, CPU or any IO device.

The critical section cannot be executed by more than one process at the same time; operating system faces the difficulties in allowing and disallowing the processes from entering the critical section.

The critical section problem is used to design a set of protocols which can ensure that the Race condition among the processes will never arise.

In order to synchronize the cooperative processes, our main task is to solve the critical section problem. We need to provide a solution in such a way that the following conditions can be satisfied.



Any solution to the critical section problem must satisfy three requirements:

- **Mutual Exclusion** : If a process is executing in its critical section, then no other process is allowed to execute in the critical section.
- **Progress** : If no process is executing in the critical section and other processes are waiting outside the critical section, then only those processes that are not executing in their remainder section can participate in deciding which will enter in the critical section next, and the selection can not be postponed indefinitely.
- **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.

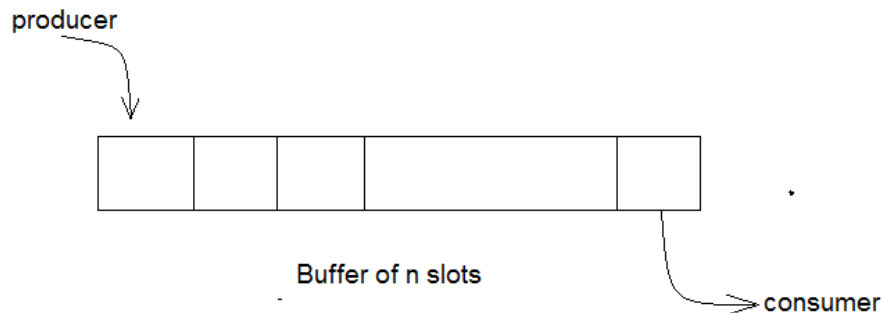
#### **Peterson's Solution**

Peterson's Solution is a classical software based solution to the critical section problem.



		<p>In Peterson's solution, we have two shared variables:</p> <ul style="list-style-type: none"> <li>boolean flag[i] : Initialized to FALSE, initially no one is interested in entering the critical section</li> <li>int turn : The process whose turn is to enter the critical section.</li> </ul> <div style="border: 1px solid black; padding: 10px; margin: 10px 0;"> <pre> do {      flag[i] = TRUE ;     turn = j ;     while (flag[j] &amp;&amp; turn == j) ;          critical section      flag[i] = FALSE ;          remainder section  } while (TRUE) ; </pre> </div> <p>Peterson's Solution preserves all three conditions :</p> <ul style="list-style-type: none"> <li>Mutual Exclusion is assured as only one process can access the critical section at any time.</li> <li>Progress is also assured, as a process outside the critical section does not block other processes from entering the critical section.</li> <li>Bounded Waiting is preserved as every process gets a fair chance.</li> </ul> <p>Disadvantages of Peterson's Solution</p> <ul style="list-style-type: none"> <li>It involves Busy waiting</li> <li>It is limited to 2 processes.</li> </ul>
	Q16	<p>a) Describe the Bounded - buffer problem and give a solution for the same using semaphores. Write the structure of producer and consumer processes.</p> <p>b) Why is deadlock state more critical than starvation? Draw the resource allocation graph (i) with deadlock (ii) with a cycle but no deadlock.</p>
	Ans	<p>a) The bounded-buffer problems (aka the producer-consumer problem) is a classic example of concurrent access to a shared resource. A bounded buffer lets multiple producers and multiple consumers share a single buffer. Producers write data to the buffer and consumers read data from the buffer.</p> <ul style="list-style-type: none"> <li>Producers must block if the buffer is full.</li> <li>Consumers must block if the buffer is empty.</li> </ul>

There is a buffer of  $n$  slots and each slot is capable of storing one unit of data. There are two processes running, namely, **producer** and **consumer**, which are operating on the buffer.



### Bounded Buffer Problem

A producer tries to insert data into an empty slot of the buffer. A consumer tries to remove data from a filled slot in the buffer. As you might have guessed by now, those two processes won't produce the expected output if they are being executed concurrently.

There needs to be a way to make the producer and consumer work in an independent manner.

### Solution:

One solution of this problem is to use semaphores. The semaphores which will be used here are:

- $m$ , a **binary semaphore** which is used to acquire and release the lock.
- $empty$ , a **counting semaphore** whose initial value is the number of slots in the buffer, since, initially all slots are empty.
- $full$ , a **counting semaphore** whose initial value is 0.

At any instant, the current value of  $empty$  represents the number of empty slots in the buffer and  $full$  represents the number of occupied slots in the buffer.

### The Producer Operation

The pseudocode of the producer function looks like this:

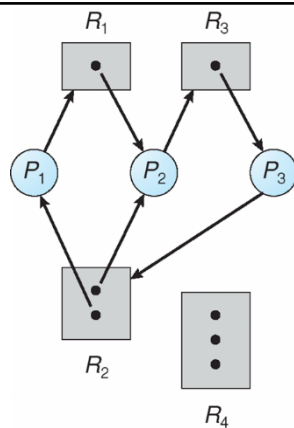
do

{

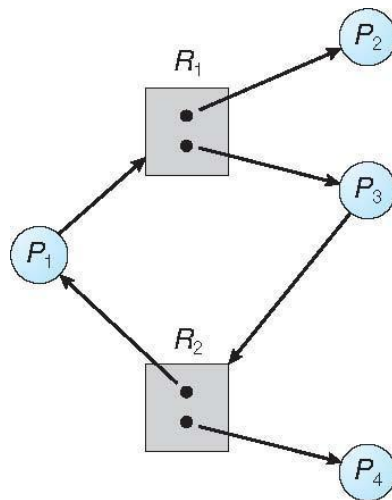
// wait until  $empty > 0$  and then decrement 'empty'

	<pre> wait(empty); // acquire lock wait(mutex);  /* perform the insert operation in a slot */  // release lock signal(mutex); // increment 'full' signal(full); } while(TRUE) </pre> <ul style="list-style-type: none"> <li>• Looking at the above code for a producer, we can see that a producer first waits until there is atleast one empty slot.</li> <li>• Then it decrements the <b>empty</b> semaphore because, there will now be one less empty slot, since the producer is going to insert data in one of those slots.</li> <li>• Then, it acquires lock on the buffer, so that the consumer cannot access the buffer until producer completes its operation.</li> <li>• After performing the insert operation, the lock is released and the value of <b>full</b> is incremented because the producer has just filled a slot in the buffer.</li> </ul> <hr/> <p><b>The Consumer Operation</b></p> <p>The pseudocode for the consumer function looks like this:</p> <pre> do { // wait until full &gt; 0 and then decrement 'full' wait(full); // acquire the lock wait(mutex); </pre>
--	---

		<pre> /* perform the remove operation in a slot */  // release the lock signal(mutex); // increment 'empty' signal(empty); } while(TRUE); </pre> <ul style="list-style-type: none"> <li>• The consumer waits until there is atleast one full slot in the buffer.</li> <li>• Then it decrements the <b>full</b> semaphore because the number of occupied slots will be decreased by one, after the consumer completes its operation.</li> <li>• After that, the consumer acquires lock on the buffer.</li> <li>• Following that, the consumer completes the removal operation so that the data from one of the full slots is removed.</li> <li>• Then, the consumer releases the lock.</li> <li>• Finally, the <b>empty</b> semaphore is incremented by 1, because the consumer has just removed data from an occupied slot, thus making it empty.</li> </ul> <p>b) A fair system prevents starvation and deadlock. Starvation occurs when one or more threads in your program are blocked from gaining access to a resource and, as a result, cannot make progress. Deadlock, the ultimate form of starvation, occurs when two or more threads are waiting on a condition that cannot be satisfied.</p> <p><b>Resource Allocation Graph With A Deadlock</b></p>
--	--	---



**Graph With A Cycle But No Deadlock**



#### Module 4

Q17

- Explain with the help of supporting diagram, how translation look-aside buffer (TLB) improves the performance of a paging system.
- With a diagram write the steps involved in handling a page fault

Ans

- In Operating System (Memory Management Technique : Paging), for each process page table will be created, which will contain Page Table Entry (PTE). This PTE will contain information like frame number (The address of main memory where we want to refer), and some other useful bits (e.g., valid/invalid bit, dirty bit, protection bit etc). This page table entry (PTE) will tell where in the main memory the actual page is residing.

Now the question is where to place the page table, such that overall access time (or reference time) will be less.

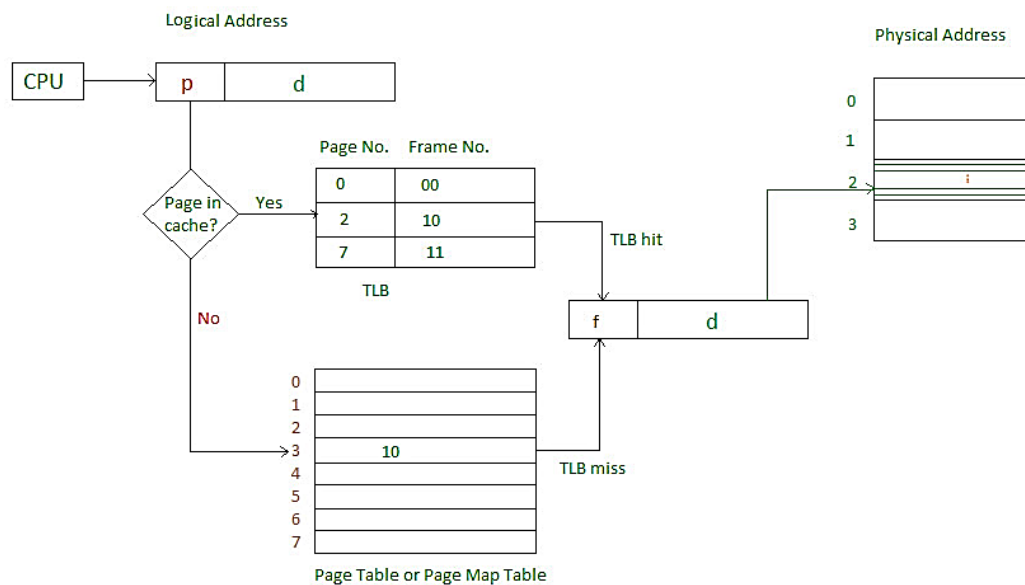
The problem initially was to fast access the main memory content based on address generated by CPU (i.e logical/virtual address). Initially, some people thought of using registers to store page table, as they are high-speed memory so access time will be less.

The idea used here is, place the page table entries in registers, for each request generated from CPU (virtual address), it will be matched to the appropriate page number of the page table, which will now tell where in the main memory that corresponding page resides. Everything seems right here, but the problem is register size is small (in practical, it can accommodate maximum of 0.5k to 1k page table entries) and process size may be big hence the required page table will also be big (lets say this page table contains 1M entries), so registers may not hold all the PTE's of Page table. So this is not a practical approach.

To overcome this size issue, the entire page table was kept in main memory. but the problem here is two main memory references are required:

1. To find the frame number
2. To go to the address specified by frame number

To overcome this problem a high-speed cache is set up for page table entries called a Translation Lookaside Buffer (TLB). Translation Lookaside Buffer (TLB) is nothing but a special cache used to keep track of recently used transactions. TLB contains page table entries that have been most recently used. Given a virtual address, the processor examines the TLB if a page table entry is present (TLB hit), the frame number is retrieved and the real address is formed. If a page table entry is not found in the TLB (TLB miss), the page number is used as index while processing page table. TLB first checks if the page is already in main memory, if not in main memory a page fault is issued then the TLB is updated to include the new page entry.



#### Steps in TLB hit:

1. CPU generates virtual (logical) address.

2. It is checked in TLB (present).
3. Corresponding frame number is retrieved, which now tells where in the main memory page lies.

#### Steps in TLB miss:

1. CPU generates virtual (logical) address.
2. It is checked in TLB (not present).
3. Now the page number is matched to page table residing in main memory (assuming page table contains all PTE).
4. Corresponding frame number is retrieved, which now tells where in the main memory page lies.
5. The TLB is updated with new PTE (if space is not there, one of the replacement technique comes into picture i.e either FIFO, LRU or MFU etc).

**Effective memory access time(EMAT) :** TLB is used to reduce effective memory access time as it is a high speed associative cache.

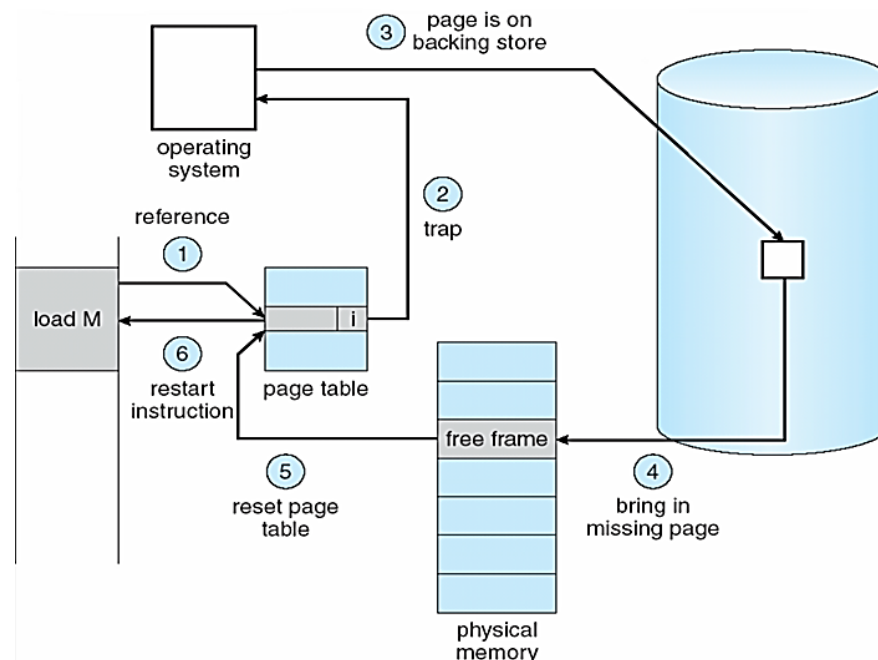
$$\text{EMAT} = h*(c+m) + (1-h)*(c+2m)$$

where, h = hit ratio of TLB

m = Memory access time

c = TLB access time

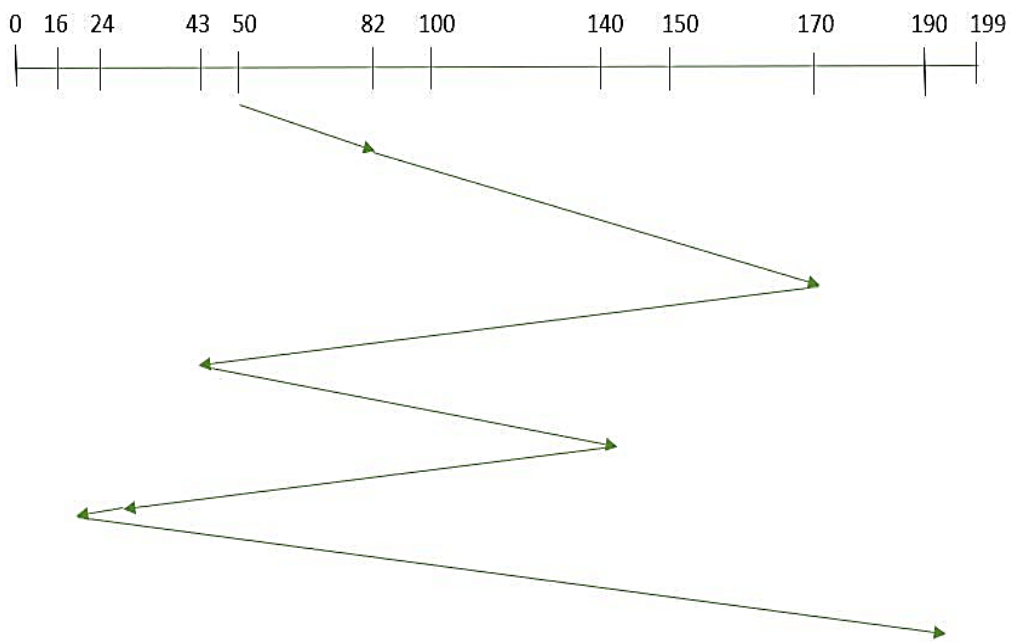
#### b) Handling the page fault:



1. Check the location of the referenced page in the PMT
2. If a page fault occurred, call on the operating system to fix it





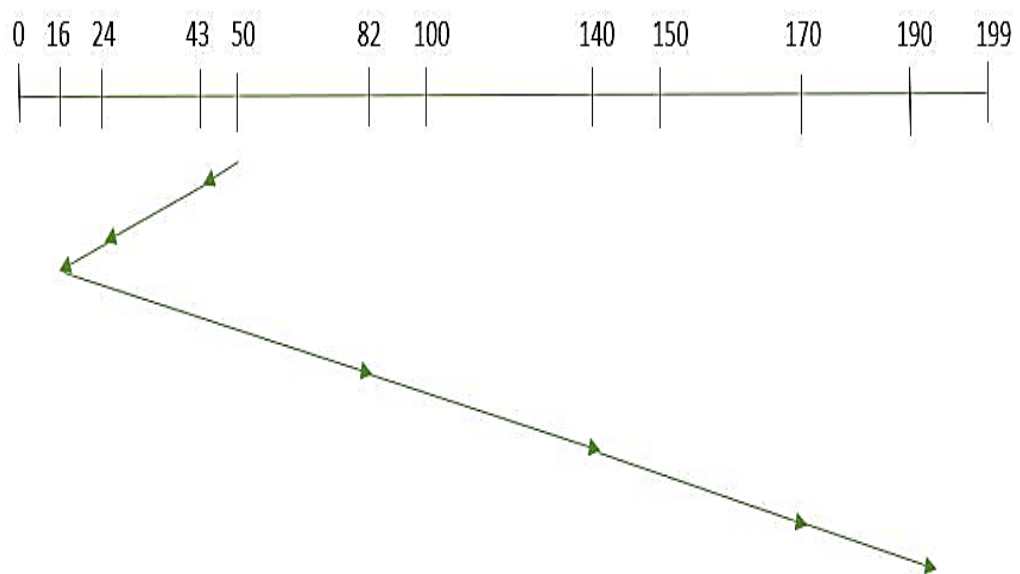
		<p>15 faults</p> <p>b) (i) 1850 (ii) 800 (iii) 2670 (iv) 125 (v) 3830</p>
Module 5		
	Q19	<p>a) Explain FCFS, SSTF and SCAN disk scheduling algorithms, using the given disk queue of requests: 82,170,43,140,24,16,190. Find the total seek time for each case. Assume that, the disk has 200 cylinders ranging from 0 to 199 and the current position of head is at cylinder 50.</p> <p>b) Explain indexed allocation method with an example.</p>
	Ans	<p>a) <b><u>Disk Scheduling Algorithms</u></b></p> <p>1. <b><u>FCFS:</u></b> FCFS is the simplest of all the Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue. Let us understand this with the help of an example.</p> <p><u>Example:</u>          Suppose the order of request is- (82,170,43,140,24,16,190)          And current position of Read/Write head is : 50</p>  <p>So, total seek time:  <math display="block">= (82-50) + (170-82) + (170-43) + (140-43) + (140-24) + (24-16) + (190-16)</math> <math display="block">= 642</math></p> <p>Advantages:</p> <ul style="list-style-type: none"> <li>• Every request gets a fair chance</li> <li>• No indefinite postponement</li> </ul> <p>Disadvantages:</p>

- Does not try to optimize seek time
  - May not provide the best possible service
2. **SSTF:** In SSTF (Shortest Seek Time First), requests having shortest seek time are executed first. So, the seek time of every request is calculated in advance in the queue and then they are scheduled according to their calculated seek time. As a result, the request near the disk arm will get executed first. SSTF is certainly an improvement over FCFS as it decreases the average response time and increases the throughput of system. Let us understand this with the help of an example.

Example:

Suppose the order of request is- (82,170,43,140,24,16,190)

And current position of Read/Write head is : 50



So, total seek time:

$$= (50-43) + (43-24) + (24-16) + (82-16) + (140-82) + (170-140) + (190-170) \\ = 208$$

Advantages:

- Average Response Time decreases
- Throughput increases

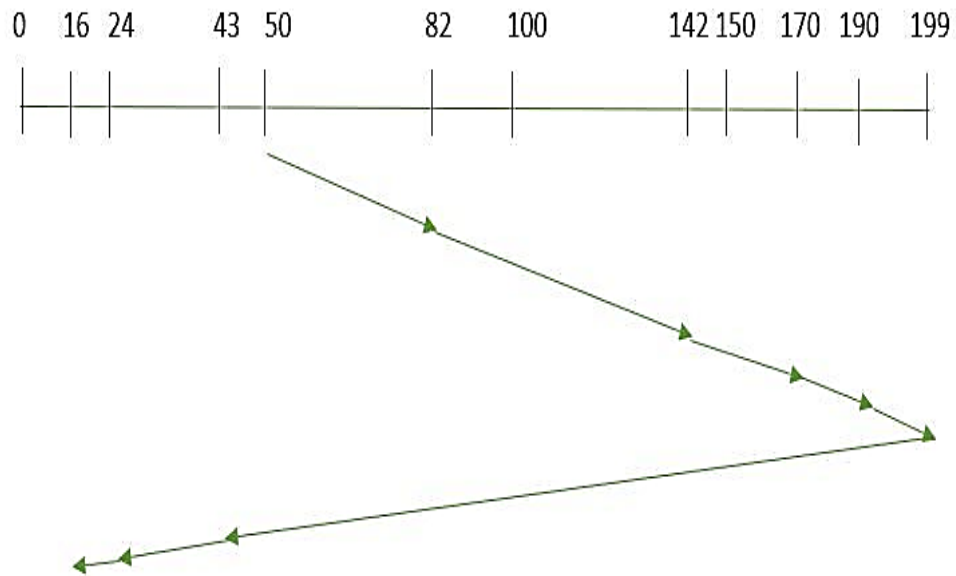
Disadvantages:

- Overhead to calculate seek time in advance
  - Can cause Starvation for a request if it has higher seek time as compared to incoming requests
  - High variance of response time as SSTF favours only some requests
3. **SCAN:** In SCAN algorithm the disk arm moves into a particular direction and services the requests coming in its path and after reaching the end of disk, it reverses its direction and again services the request arriving in its path. So, this algorithm

works as an elevator and hence also known as **elevator algorithm**. As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.

Example:

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move “**towards the larger value**”.



Therefore, the seek time is calculated as:

$$\begin{aligned} &= (199 - 50) + (199 - 16) \\ &= 332 \end{aligned}$$

Advantages:

- High throughput
- Low variance of response time
- Average response time

Disadvantages:

- Long waiting time for requests for locations just visited by disk arm

b) The allocation methods define how the files are stored in the disk blocks. There are three main disk space or file allocation methods.

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

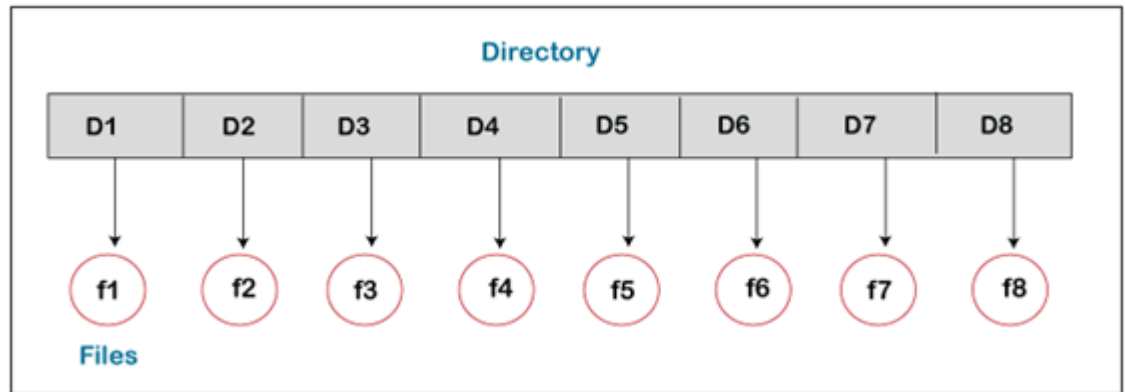
The main idea behind these methods is to provide:

- Efficient disk space utilization.

		<ul style="list-style-type: none"> <li>Fast access to the file blocks.</li> </ul> <p>Indexed Allocation method:</p> <p>In this scheme, a special block known as the <b>Index block</b> contains the pointers to all the blocks occupied by a file. Each file has its own index block. The <i>i</i>th entry in the index block contains the disk address of the <i>i</i>th file block. The directory entry contains the address of the index block as shown in the image:</p> <p><b>Advantages:</b></p> <ul style="list-style-type: none"> <li>This supports direct access to the blocks occupied by the file and therefore provides fast access to the file blocks.</li> <li>It overcomes the problem of external fragmentation.</li> </ul> <p><b>Disadvantages:</b></p> <ul style="list-style-type: none"> <li>The pointer overhead for indexed allocation is greater than linked allocation.</li> <li>For very small files, say files that expand only 2-3 blocks, the indexed allocation would keep one entire block (index block) for the pointers which is inefficient in terms of memory utilization. However, in linked allocation we lose the space of only 1 pointer per block.</li> </ul>
	Q20	<p>a) Explain the different directory structures used in file system.</p> <p>b) Different users may need different types of access to a file or directory. Explain the most general scheme to implement identity dependent access.</p>
	Ans	<p><b>a) Types of Directory Structure</b></p> <p>There are various types of directory structure:</p> <ul style="list-style-type: none"> <li>Single-Level Directory</li> <li>Two-Level Directory</li> <li>Tree-Structured Directory</li> <li>Acyclic Graph Directory</li> </ul>

- General-Graph Directory

**Single-Level Directory:** – Single-Level Directory is the easiest directory structure. There is only one directory in a single-level directory, and that directory is called a root directory. In a single-level directory, all the files are present in one directory that makes it easy to understand. In this, under the root directory, the user cannot create the subdirectories.



### Advantages of Single-Level Directory

The advantages of the single-level directory are:

The implementation of a single-level directory is so easy.

In a single-level directory, if all the files have a small size, then due to this, the searching of the files will be easy.

In a single-Level directory, the operations such as searching, creation, deletion, and updating can be performed.

### Disadvantages of Single-Level Directory

The disadvantages of Single-Level Directory are:

If the size of the directory is large in Single-Level Directory, then the searching will be tough.

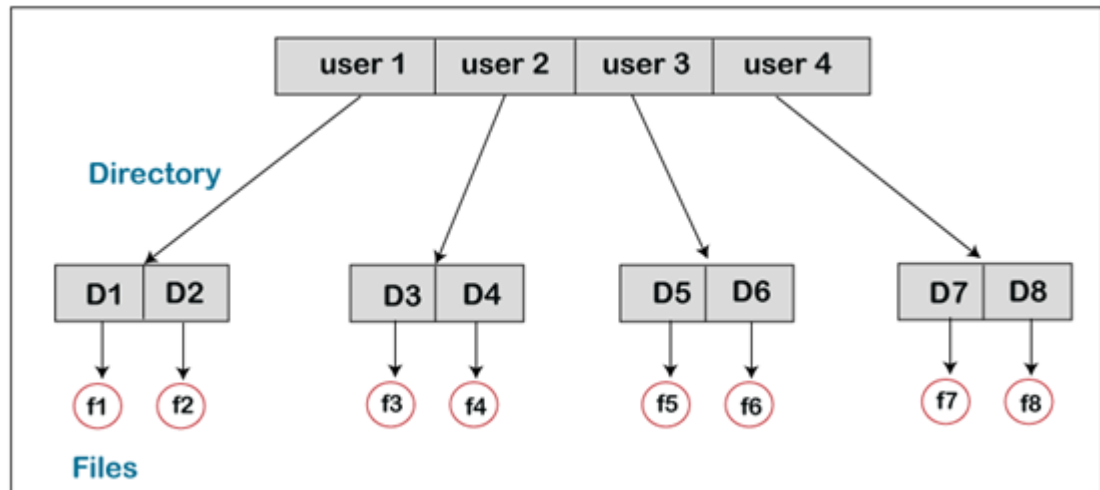
In a single-level directory, we cannot group the similar type of files.

Another disadvantage of a single-level directory is that there is a possibility of collision because the two files cannot have the same name.

The task of choosing the unique file name is a little bit complex.

### Two-Level Directory

Two-Level Directory is another type of directory structure. In this, it is possible to create an individual directory for each of the users. There is one master node in the two-level directory that include an individual directory for every user. At the second level of the directory, there is a different directory present for each of the users. Without permission, no user can enter into the other user's directory.



### Characteristics of Two-Level Directory

The characteristics of the two-level directory are:

In a two-level directory, there may be same file name of different users.

There is a pathname of each file such as /**User-name/directory-name/**

In a two-level directory, we cannot group the files which are having the same name into a single directory for a specific user.

In a two-level directory, searching is more effective because there is only one user's list, which is required to be traversed.

### Advantages of Two-Level Directory

The advantages of the two-level directory are:

In the two-level directory, various users have the same file name and also directory name.

Because of using the user-grouping and pathname, searching of files are quite easy.

### Disadvantages of Two-Level Directory

The disadvantages of the two-level directory are:

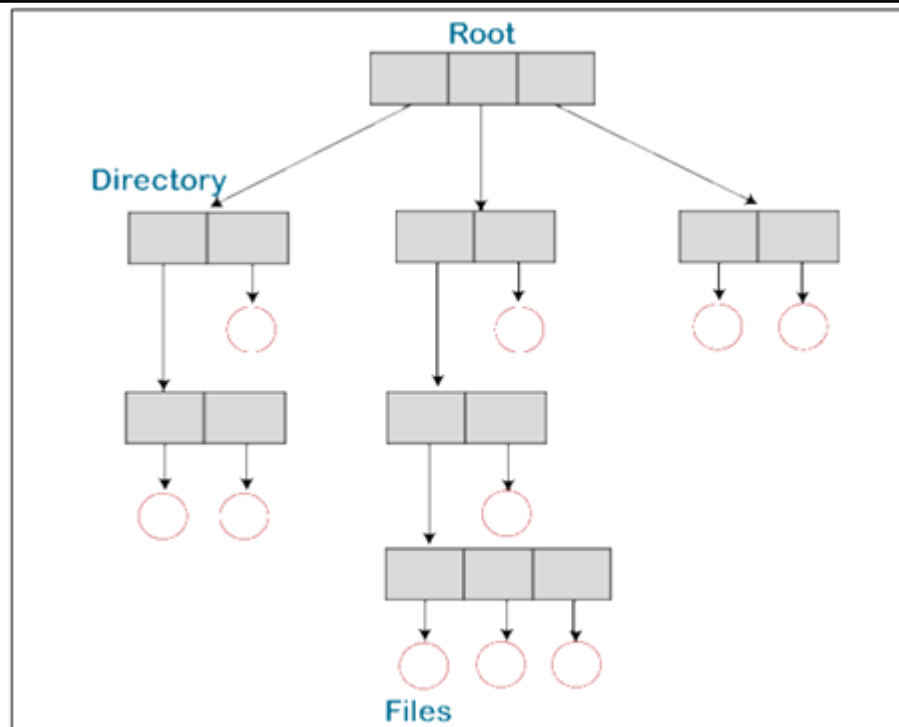
In a two-level directory, one user cannot share the file with another user.

Another disadvantage with the two-level directory is it is not scalable.

### Tree-Structured Directory

A Tree-structured directory is another type of directory structure in which the directory entry may be a sub-directory or a file. The tree-structured directory reduces the limitations of the two-level directory. We can group the same type of files into one directory.

In a tree-structured directory, there is an own directory of each user, and any user is not allowed to enter into the directory of another user. Although the user can read the data of root, the user cannot modify or write it. The system administrator only has full access to the root directory. In this, searching is quite effective and we use the current working concept. We can access the file by using two kinds of paths, either absolute or relative.



### **Advantages of tree-structured directory**

The advantages of the tree-structured directory are:

The tree-structured directory is very scalable.

In the tree-structures directory, the chances of collision are less.

In the tree-structure directory, the searching is quite easy because, in this, we can use both types of paths, which are the absolute path and relative path.

### **Disadvantages of Tree-Structure Directory**

The disadvantages of tree-structure directory are:

In the tree-structure directory, the files cannot be shared.

Tree-structure directory is not efficient because, in this, if we want to access a file, then it may go under multiple directories.

Another disadvantage of the tree-structure directory is that each file does not fit into the hierarchal model. We have to save the files into various directories.

### **Acyclic-Graph Directory**

In the tree-structure directory, the same files cannot exist in the multiple directories, so sharing the files is the main problem in the tree-structure directory. With the help of the acyclic-graph directory, we can provide the sharing of files. In the acyclic-graph directory, more than one directory can point to a similar file or subdirectory. We can share those files among the two directory entries.

With the help of aliases, and links, we can create this type of directory graph. We may also have a different path for the same file. Links may be of two kinds, which are hard link (physical) and symbolic (logical).

If we delete the files in acyclic graph structures, then

In the hard link (physical) case, we can remove the actual files only if all the references to the file are deleted.

In the symbolic link (logical) case, we just delete the file, and there is only a dangling point that is left.

### **Advantages of Acyclic-Graph Directory**

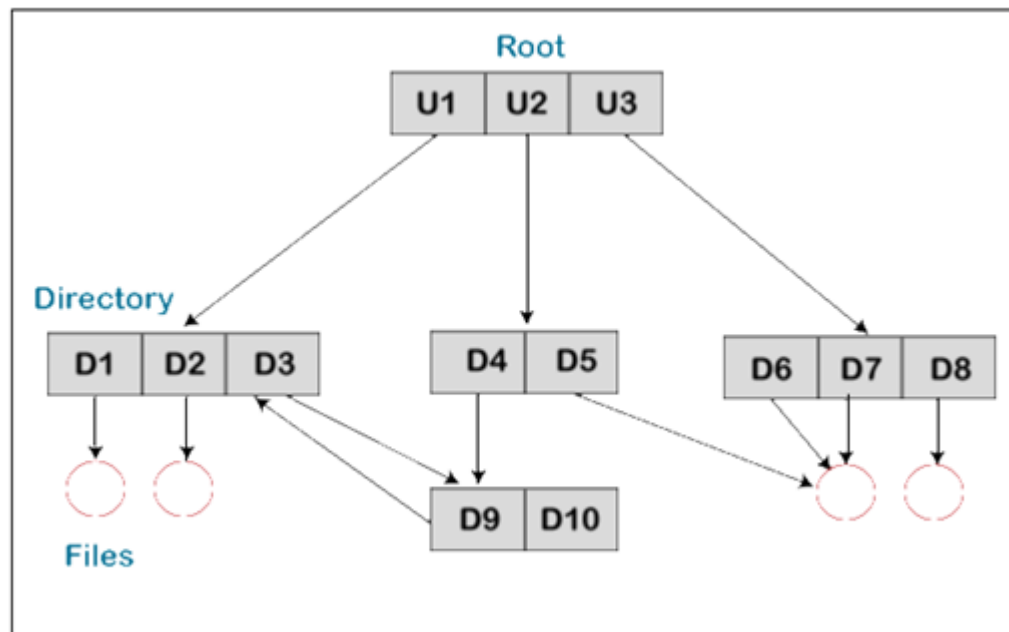
The advantages of the acyclic-graph directory are:  
In the acyclic-graph directory, the sharing of files is possible.  
In the acyclic-graph directory, because of different-different paths, searching is easy.

### **Disadvantages of Acyclic-Graph Directory**

The disadvantages of acyclic-graph directory are:  
If the files are shared through linking, there may be a problem in the case of deleting.  
If we are using softlink, then in this case, if the file is deleted then there is only a dangling pointer which is left.  
If we are using hardlink, in this case, when we delete a file, then we also have to remove all the reference connected with it.

### **General-Graph Directory**

The General-Graph directory is another vital type of directory structure. In this type of directory, within a directory we can create cycle of the directory where we can derive the various directory with the help of more than one parent directory.  
The main issue in the general-graph directory is to calculate the total space or size, taken by the directories and the files.



### **Advantages of General-Graph directory**

The advantages of general-graph directory are:  
The General-Graph directory is more flexible than the other directory structure.  
Cycles are allowed in the general-graph directory.

### **Disadvantages of General-Graph Directory**

The disadvantages of general-graph directory are:  
In general-graph directory, garbage collection is required.  
General-graph directory is more costly, among other directory structures.



b)

The need to protect files is a direct result of the ability to access files. Systems that do not permit access to the files of other users do not need protection. Thus, we could provide complete protection by prohibiting access. Alternatively, we could provide free access with no protection. Both approaches are too extreme for general use. What is needed is controlled access. Protection mechanisms provide controlled access by limiting the types of file access that can be made. Access is permitted or denied depending on several factors, one of which is the type of access requested. Several different types of operations may be controlled:

- Read. Read from the file.
- Write. Write or rewrite the file.
- Execute. Load the file into memory and execute it.
- Append. Write new information at the end of the file.
- Delete. Delete the file and free its space for possible reuse.
- List. List the name and attributes of the file.

Other operations, such as renaming, copying, and editing the file, may also be controlled. For many systems, however, these higher-level functions may be implemented by a system program that makes lower-level system calls. Protection is provided at only the lower level. For instance, copying a file may be implemented simply by a sequence of read requests. In this case, a user with read access can also cause the file to be copied, printed, and so on. Many protection mechanisms have been proposed. Each has advantages and disadvantages and must be appropriate for its intended application. A small computer system that is used by only a few members of a research group, for example, may not need the same types of protection as a large corporate computer that is used for research, finance, and personnel operations.

#### **Access Control**

The most common approach to the protection problem is to make access dependent on the identity of the user. Different users may need different types of access to a file or directory. The most general scheme to implement identity dependent access is to associate with each file and directory an access-control list (ACL) specifying user names and the types of access allowed for each user.

When a user requests access to a particular file, the operating system checks the access list associated with that file. If that user is listed for the requested access, the access is allowed.

Otherwise, a protection violation occurs, and the user job is denied access to the file.

This approach has the advantage of enabling complex access methodologies. The main problem with access lists is their length. If we want to allow everyone to read a file, we must list all users with read access. This technique has two undesirable consequences:

- Constructing such a list may be a tedious and unrewarding task, especially if we do not know in advance the list of users in the system.
- The directory entry, previously of fixed size, now must be of variable size, resulting in more complicated space management.

These problems can be resolved by use of a condensed version of the access list. To condense the length of the access-control list, many systems recognize three classifications of users in connection with each file:

- Owner. The user who created the file is the owner.

- Group. A set of users who are sharing the file and need similar access is a group, or work group.
- Universe. All other users in the system constitute the universe.

### **The Permission Indicators**

While using `ls -l` command, it displays various information related to file permission as follows –

```
$ls -l /home/amrood
```

```
-rwxr-xr-- 1 amrood users 1024 Nov 2 00:10 myfile
```

```
drwxr-xr-- 1 amrood users 1024 Nov 2 00:10 mydir
```

Here, the first column represents different access modes, i.e., the permission associated with a file or a directory.

The permissions are broken into groups of threes, and each position in the group denotes a specific permission, in this order: read (r), write (w), execute (x) –

1. The first three characters (2-4) represent the permissions for the file's owner. For

example, `-rwxr-xr--` represents that the owner has read (r), write (w) and execute (x) permission.

2. The second group of three characters (5-7) consists of the permissions for the group to

which the file belongs. For example, `-rwxr-xr--` represents that the group has read (r) and execute (x) permission, but no write permission.

3. The last group of three characters (8-10) represents the permissions for everyone else.

For example, `-rwxr-xr--` represents that there is read (r) only permission.

### **File Access Modes**

The permissions of a file are the first line of defense in the security of a Unix system. The basic building blocks of Unix permissions are the read, write, and execute permissions, which have been described below –

Read: Grants the capability to read, i.e., view the contents of the file.

Write: Grants the capability to modify, or remove the content of the file.

Execute: User with execute permissions can run a file as a program.

### **Directory Access Modes**

Directory access modes are listed and organized in the same manner as any other file.

There are a few differences that need to be mentioned –

Read : Access to a directory means that the user can read the contents. The user can look at the filenames inside the directory.

Write : Access means that the user can add or delete files from the directory.

Execute: Executing a directory doesn't really make sense, so think of this as a traverse permission. A user must have execute access to the bin directory in order to execute the `ls` or the `cd` command.

### **Changing Permissions**

To change the file or the directory permissions, you use the `chmod` (change mode) command.

There are two ways to use `chmod` — the symbolic mode and the absolute mode.

