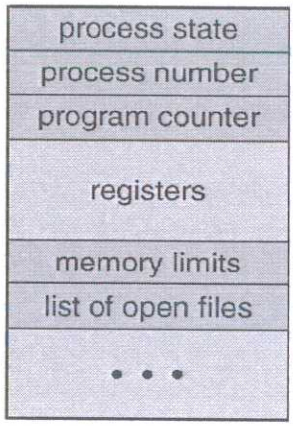
Scoring Indicators

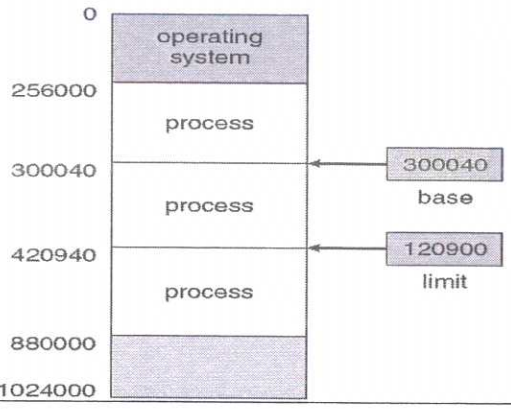
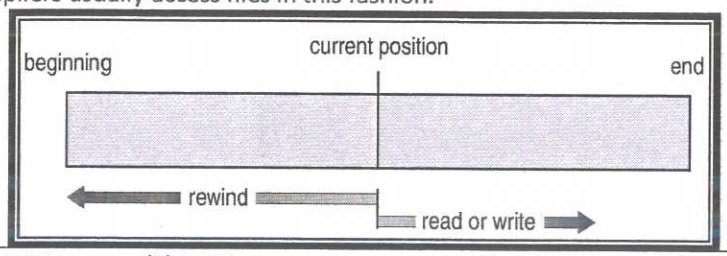
Code B

Course Code: **(21) 5132** Course: **Operating System**

<table>
<tr><th>Qn. No.</th><th>Key</th><th>Marks division</th><th>Total Marks</th></tr>
<tr><td></td><td>Part A</td><td></td><td></td></tr>
<tr><td>I</td><td></td><td></td><td></td></tr>
<tr><td>1</td><td>true</td><td>1</td><td>1</td></tr>
<tr><td>2</td><td>Executable code</td><td>1</td><td>1</td></tr>
<tr><td>3</td><td>Aging</td><td>1</td><td>1</td></tr>
<tr><td>4</td><td>A critical section is a code segment that can be accessed by only one process at a time. The critical section contains shared variables that need to be synchronized to maintain the consistency of data variables.</td><td>1</td><td>1</td></tr>
<tr><td>5</td><td>A process is an instance of the program that is a process is started when a program is executed.</td><td>1</td><td>1</td></tr>
<tr><td>6</td><td>user processes</td><td>1</td><td>1</td></tr>
<tr><td>7</td><td>true</td><td>1</td><td>1</td></tr>
<tr><td>8</td><td>First Come First Served (FCFS)</td><td>1</td><td>1</td></tr>
<tr><td>9</td><td>One bit in each directory entry defines the entry as a file (0) or as a subdirectory (1).</td><td>1</td><td>1</td></tr>
<tr><td></td><td>Part B</td><td></td><td></td></tr>
<tr><td>II</td><td></td><td></td><td></td></tr>
<tr><td>1</td><td>
<table>
<tr><th>System Software (SS)</th><th>Application Software (AS)</th></tr>
<tr><td>SS is the type of software which is the interface between AS and system.</td><td>AS is the type of software which runs as per user request. It runs on the platform which is provided by SS.</td></tr>
<tr><td>SS are essential for operating the hardware. Without this software, a computer even may not start or function properly.</td><td>AS are not essential for the operation of the computer. These are installed as per the user's requirements.</td></tr>
<tr><td>SS is used for operating computer hardware.</td><td>AS is used by user to perform specific task.</td></tr>
<tr><td>SS are installed on the computer when OS is installed.</td><td>AS are installed according to user's requirements.</td></tr>
<tr><td>SS are specific to hardware, so less or no user interaction available in case of SS.</td><td>Users can interact with an AS with the help of a User Interface (UI).</td></tr>
<tr><td>SS can run independently. It provides platform for running AS.</td><td>An AS cannot run independently. It cannot run without the presence of SS.</td></tr>
<tr><td>Examples of SS include OSs, compilers, assemblers, debuggers, drivers, etc.</td><td>Examples of AS include word processors, web browsers, media players, etc.</td></tr>
</table>
</td><td>Any three
1 x 3</td><td>3</td></tr>
</table>

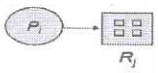
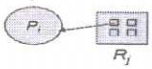
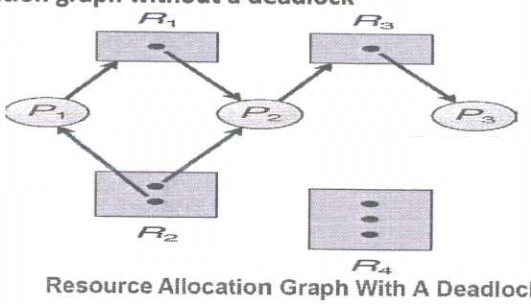
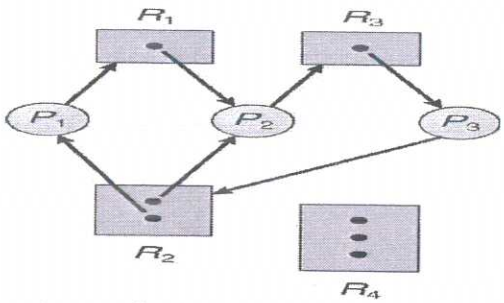
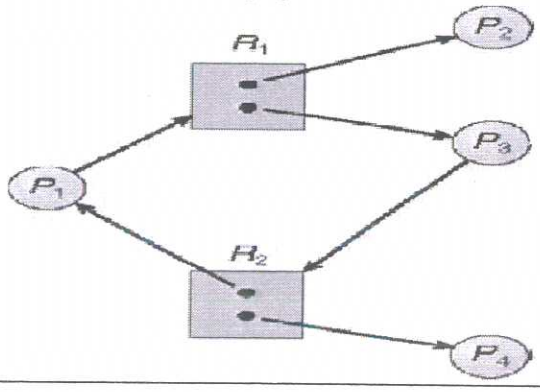
2	**Advantages of Timesharing OS:** • It helps to reduce the CPU idle time. • It offers the benefits of a fast response. • It avoids the duplication of software. • Each job gets an equal opportunity. **Disadvantages of TOS:** • Data communication happens in the timesharing OS. • It has the problem of reliability.	**Any three 1 x 3**	**3**
3	Information associated with each process (also called **task control block**) □ Process state – running, waiting, etc □ Program counter – location of instruction to next execute □ CPU registers – contents of all process-centric registers □ CPU scheduling information- priorities, scheduling queue pointers □ Memory-management information – memory allocated to the process □ Accounting information – CPU used, clock time elapsed since start, time limits □ I/O status information – I/O devices allocated to process, list of open files 	**Any six 0.5 x 6**	**3**
4	Deadlock can arise if four conditions hold simultaneously. • **Mutual exclusion:** only one process at a time can use a resource • **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes • **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task • **Circular wait:** there exists a set {P0, P1, ..., Pn} of waiting processes such that P0 is waiting for a resource that is held by P1, P1 is waiting for a resource that is held by P2, ..., Pn–1 is waiting for a resource that is held by Pn, and Pn is waiting for a resource that is held by P0.	**3**	**3**
5	Address binding of instructions and data to memory addresses can happen at three different stages: **Compile time**: If memory location known a priori, absolute code can be generated; must recompile code if starting location changes. **Load time**: Must generate relocatable code if memory location is not known at compile time. **Execution time**: Binding delayed until run time if the process can be moved during its execution from one memory segment to another.	**1 x 3**	**3**
6	• **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used. Systems with fixed-sized allocation units, such as the single-partition scheme and paging, suffer from internal fragmentation. Solutions: Best fit allocation and dynamic partitioning	**3**	**3**

7	**Contiguous memory allocation:**
	• Main memory must support both OS and user processes
	• Limited resource, must allocate efficiently
	• Contiguous allocation is one early method
	  ○ Main memory usually into two **partitions**:
	  ▪ Resident operating system, usually held in low memory with interrupt vector
	  ▪ User processes then held in high memory
	  ▪ Each process contained in single contiguous section of memory
	• Relocation registers used to protect user processes from each other, and from changing operating-system code and data
	  • Base register contains value of smallest physical address
	  • Limit register contains range of logical addresses – each logical address must be less than the limit register
	  • MMU maps logical address *dynamically*
		3	3

8	• A pair of **base** and **limit registers** define the logical address space • CPU must check every memory access generated in user mode to be sure it is between base and limit for that user. • Example:  	3	3
9	**Sequential Access** • The simplest access method is sequential access. • Refers to reading or writing data records in sequential order, that is, one record after the other. To read record 10, for example, we would first need to read records 1 through 9 • Information in the file is processed in order, one record after the other. • This mode of access is by far the most common; for example, editors and compilers usually access files in this fashion.  	3	3
10	(i) FCFS, (ii) SSTF, (iii) Scan, (iv) C-Scan, (v) Look, and (vi) C-Look.	0.5 x 6	3

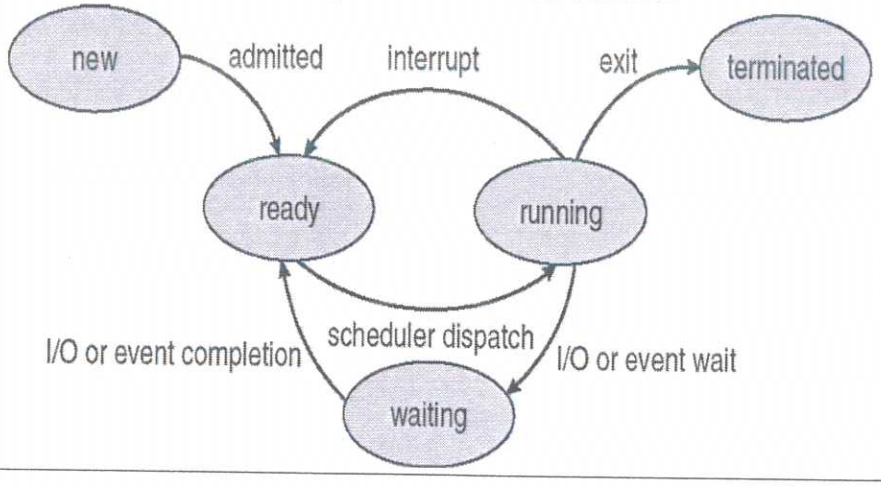
	Part C		
III	**The Functions of OS:** 1) Process Management, 2) I/O Device Management, 3) File Management, 4) Network Management, 5) Main Memory Management, 6) Secondary Storage Management, 7) Security Management, 8) Command Interpreter System, 9) Control over system performance, 10) Job Accounting, 11) Error Detection and Correction, 12) Resources allocation, 13) Information and Resource Protection, 14) Monitoring activities, 15) Handling the I/O Operations, 16) Special Control Program, 17) Job Priority, 18) Coordination between other software and user	**Any fourteen 0.5 x 14**	**7**
IV	**Real Time Operating System (RTOS):** A RTOS is a type of OS designed to serve real-time applications that process data as it arrives. It completes a task within a specific time. The logical result of computation and the time required to produce the result determine the correctness of the system output. It includes methods for real-time task scheduling. It is primarily used on embedded systems. It is highly useful for timing applications or activities that are performed within a particular time limit. It uses strict time limits to drive task execution in an external environment. • RTOSs require accurate results and timely results, which means that the results must be produced within a certain time limit, or the system will fail. It is primarily used in control device applications like automobile-engine fuel injection systems, industrial control systems, weapon systems, medical imaging systems, etc. **Advantages (any 2):** • A RTOS often takes less time to shift from one task to another. • An RTOS is a system that is available 24/7 because it produces maximum results. • RTOSs, particularly those based on hard RTOS, are completely error-free. • An RTOS ensures that the system consumes more resources while keeping all devices active. • A RTOS focuses on one application at a time. **Disadvantages (any 2):** • A RTOS constantly experiences signal interruptions. • An RTOS focuses on only one application at a time. It is used to maintain accuracy and reduce errors. All other low-priority applications need to be on waiting. • Although a RTOS can focus on specific applications, it is not the same as multitasking. • Program crashes may often be experienced while using a RTOS. • Complex algorithms are behind an RTOS interface.	**Expln: 3 + Advs: 2 + Disadvs: 2**	**7**

<table>
<tr>
<td>V</td>
<td>Process synchronization is the coordination of execution of multiple processes in a multi-process system to ensure that they access shared resources in a controlled and predictable manner. It aims to resolve the problem of race conditions and other synchronization issues in a concurrent system. The main objective of process synchronization is to ensure that multiple processes access shared resources without interfering with each other, and to prevent the possibility of inconsistent data due to concurrent access. To achieve this, various synchronization techniques such as semaphores, monitors, and critical sections are used.
Advantages of Process Synchronization (any 3):
• Ensures data consistency and integrity
• Avoids race conditions
• Prevents inconsistent data due to concurrent access
• Supports efficient and effective use of shared resources
Disadvantages of Process Synchronization (any 3):
• Adds overhead to the system
• Can lead to performance degradation
• Increases the complexity of the system
• Can cause deadlocks if not implemented properly.</td>
<td>Defn: 1 +
Adv: 3 +
Disadv: 3</td>
<td>7</td>
</tr>
<tr>
<td>VI</td>
<td>First Come First Serve (FCFS): It doesn't include any complex logic, it just puts the process requests in a queue and executes it one by one. FCFS is pretty simple and easy to implement. Every process will get a chance to run, so starvation doesn't occur. There is no option for pre-emption of a process. If a process is started, then CPU executes the process until it ends. Because there is no pre-emption, if a process executes for a long time, the processes in the back of the queue will have to wait for a long time before they get a chance to be executed.
Shortest Job First (SJF): Short processes are executed first and then followed by longer processes. The throughput is increased because more processes can be executed in less amount of time. The time taken by a process must be known by the CPU beforehand, which is not possible. Longer processes will have more waiting time, they'll suffer starvation.
Round Robin (RR): Each process is served by the CPU for a fixed time quantum, so all processes are given the same priority. Starvation doesn't occur because for each round robin cycle, every process is given a fixed time to execute. No process is left behind. The throughput in RR largely depends on the choice of the length of the time quantum. If time quantum is longer than needed, it tends to exhibit the same behavior as FCFS. If time quantum is shorter than needed, the number of times that CPU switches from one process to another process increases. This leads to decrease in CPU efficiency.
<table>
<tr><th rowspan="2">Process ID</th><th colspan="3">WAITING TIME (ms)</th></tr>
<tr><th>FCFS</th><th>SJF</th><th>Round Robin</th></tr>
<tr><td>P_1</td><td>0</td><td>16</td><td>12</td></tr>
<tr><td>P_2</td><td>10</td><td>0</td><td>5</td></tr>
<tr><td>P_3</td><td>12</td><td>8</td><td>17</td></tr>
<tr><td>P_4</td><td>20</td><td>2</td><td>20</td></tr>
<tr><td>Avg Waiting Time</td><td>10.5</td><td>6.5</td><td>13.5</td></tr>
</table>
SJF has least average waiting time.</td>
<td>Comp: 3 +
Eg.: 3
AWT? : 1</td>
<td>7</td>
</tr>
</table>

VII	☐ Process  ☐ Resource Type with 4 instances  ☐ P_i requests instance of R_j  ☐ P_i is holding an instance of R_j  ☐ If graph contains no cycles ⇒ no deadlock ☐ If graph contains a cycle ⇒ ☐ if only one instance per resource type, then deadlock ☐ if several instances per resource type, possibility of deadlock **Resource allocation graph without a deadlock**  Resource Allocation Graph With A Deadlock  **Resource allocation graph with a circle but without a deadlock** 	**With deadlock: 4 + Without deadlock: 3**	**7**

VIII

- As a process executes, it changes state
  - **new:** The process is being created
  - **running:** Instructions are being executed
  - **waiting:** The process is waiting for some event to occur
  - **ready:** The process is waiting to be assigned to a processor
  - **terminated:** The process has finished execution



Expln: 3 + Diagram: 4 | **7**

IX

Dynamic memory allocation strategies:
How to satisfy a request of size *n* from a list of free holes?
First-fit: *Allocate the ***first*** hole that is big enough.
Best-fit: *Allocate the ***smallest*** hole that is big enough; must search entire list, unless ordered by size. *Produces the smallest leftover hole.
Worst-fit: *Allocate the ***largest*** hole; must also search entire list. *Produces the largest leftover hole.
First-fit and best-fit are better than worst-fit in terms of speed and storage utilization.

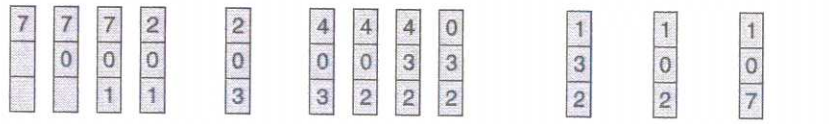
Example:

- Assuming the list of the memory holes are 10K, 4K, 20K, 18K, 7K, 9K, 12K, and 15K, with the external segmentation as 10K + 4K + 20K + 18K + 7K + 9K + 12K + 15K = 95K;
- For a successive segment requests of 15K, 5K, and 10K, the corresponding memory allocations based on the different policies can be presented in the following table:

Request/Policy	First Fit	Best Fit	Worst Fit
15K	20K	15K	20K
5K	10K	7K	18K
10K	18K	10K	15K

All the allocations become same when the blocks are of fixed (same) size.

Expln: 3 + Eg.: 3 + Condn: 1 | **7**

X	**LRU Page Replacement** If the optimal algorithm is not feasible, perhaps an approximation of the optimal algorithm is possible. The key distinction between the FIFO and OPT algorithms (other than looking backward versus forward in time) is that the FIFO algorithm uses the time when a page was brought into memory, whereas the OPT algorithm uses the time when a page is to be ***used.*** 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. We can think of this strategy as the optimal page-replacement algorithm looking backward in time, rather than forward. The LRU policy is often used as a page-replacement algorithm and is considered to be good. The major problem is ***how*** to implement LRU replacement. An LRU page-replacement algorithm may require substantial hardware assistance. The problem is to determine an order for the frames defined by the time of last use. reference string 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1  page frames Like optimal replacement, LRU replacement does not suffer from Belady's anomaly. Both belong to a class of page-replacement algorithms, called **stack algorithms** that can never exhibit Belady's anomaly.	**Expln: 3 + Eg.: 4**	**7**
XI	**Paging:** The basic method for implementing paging involves breaking physical memory into fixed-sized blocks called **frames** and breaking logical memory into blocks of the same size called **pages**. When a process is to be executed, its pages are loaded into any available memory frames from their source. Paging is a storage mechanism used to retrieve processes from the secondary storage into the main memory in the form of pages. **Demand Paging:** It is a strategy is to load pages only as they are needed. It is commonly used in virtual memory systems. With demand-paged virtual memory, pages are loaded only when they are demanded during program execution. Pages that are never accessed are thus never loaded into physical memory. A demand-paging system is similar to a paging system with swapping where processes reside in secondary memory (usually a disk). When we want to execute a process, we swap it into memory. Rather than swapping the entire process into memory, though, we use a lazy swapper. A lazy swapper never swaps a page into memory unless that page will be needed. A swapper manipulates entire processes, whereas a **pager** is concerned with the individual pages of a process. We thus use "pager," rather than "swapper," in connection with demand paging. When a process is to be swapped in, the pager guesses which pages will be used before the process is swapped out again. Instead of swapping in a whole process, the pager brings only those pages into memory. Thus, it avoids reading into memory pages that will not be used anyway, decreasing the swap time and the amount of physical memory needed. With this scheme, we need some form of hardware support to distinguish between the pages that are in memory and the pages that are on the disk.	**Paging: 2 + Demand paging: 5**	**7**

XII

The **logical address** is a virtual address created by the CPU of the computer system. The logical address of a program is generated when the program is running. A group of several logical addresses is referred to a **logical address space**. The logical address is basically used as a reference to access the physical memory locations.

A hardware device named **memory management unit** (MMU) is used to map the logical address to its corresponding physical address. However, the logical address of a program is visible to the computer user.

The **physical address** of a computer program is one that represents a location in the memory unit of the computer. The physical address is not visible to the computer user. The MMU of the system generates the physical address for the corresponding logical address.

The physical address is accessed through the corresponding logical address because a user cannot directly access the physical address. For running a computer program, it requires a physical memory space. Therefore, the logical address has to be mapped with the physical address before the execution of the program.

The concept of a logical address space that is bound to a separate **physical address space** is central to proper memory management:

Logical address – generated by the CPU; also referred to as **virtual address**

Physical address – address seen by the memory unit

Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme

Logical address space is the set of all logical addresses generated by a program. **Physical address space** is the set of all physical addresses generated by a program

Logical address improves system performance and management.

S. No.	Logical Address	Physical Address
1.	This address is generated by the CPU.	This address is a location in the memory unit.
2.	The address space consists of the set of all logical addresses.	This address is a set of all physical addresses that are mapped to the corresponding logical addresses.
3.	These addresses are generated by CPU with reference to a specific program.	It is computed using Memory Management Unit (MMU).
4.	The user has the ability to view the logical address of a program.	The user can't view the physical address of program directly.
5.	The user can use the logical address in order to access the physical address.	The user can indirectly access the physical address.

LA: 2
PA: 2
LAS: 1
PAS: 1
Which?:1

7

<table>
<tr>
<td>XIII</td>
<td>
<table>
<tr><th></th><th>Contiguous Method</th><th>Linked Allocation</th><th colspan="2">Indexed Allocation</th></tr>
<tr><td>1. Pre allocation</td><td>Necessary</td><td>Possible</td><td colspan="2">Possible</td></tr>
<tr><td>2. Fixed or variable size portions.</td><td>Variable</td><td>Fixed</td><td>Fixed</td><td>Variable</td></tr>
<tr><td>3. Portion size</td><td>Large</td><td>Small</td><td>Small</td><td>Medium</td></tr>
<tr><td>4. Allocation frequency</td><td>Once</td><td>Low to high</td><td>High</td><td>Low</td></tr>
<tr><td>5. Table size</td><td>One entry</td><td>One entry</td><td>Large</td><td>Medium</td></tr>
<tr><td>6. Access type</td><td>Random access</td><td>Direct access</td><td colspan="2">Direct access</td></tr>
<tr><td>7. Fragmentation</td><td>External</td><td>No external fragmentation</td><td colspan="2">NO</td></tr>
</table>
</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>XIV</td>
<td>
File Attributes:
• Name
• Identifier
• Type
• Location
• Size
• Protection
• Time, date, and user identification (any four)

File Operations:
• Creating a file
• Writing a file
• Reading a file
• Repositioning within a file or seek
• Deleting a file
• Truncating a file
• Appending a file
• Merging two files
• Renaming a file
• Moving a file
• Opening a file
• Closing a file
• Get attribute operation
• Set attribute operation
• Searching a file
etc (any ten)
</td>
<td>Defn: 2 + Opns: 5</td>
<td>7</td>
</tr>
</table>