

Register No.

--	--	--	--	--	--	--	--

BTech Degree Examination May 2023

Fifth Semester

Information Technology

20ITT52 – OPERATING SYSTEMS

(Regulations 2020)

Time: Three hours

Maximum: 100 marks

Answer all Questions

Part – A ($10 \times 2 = 20$ marks)

1. Mention the role of OS in computer system. [CO1,K1]
2. Differentiate between linkers and loaders. [CO1,K2]
3. Draw the process state diagram. [CO2,K1]
4. Compare preemptive and nonpreemptive scheduling. [CO2,K2]
5. Define spin lock. [CO3,K1]
6. Name the four deadlock prevention techniques. [CO3,K1]
7. What is called as internal fragmentation and external fragmentation? [CO4,K1]
8. Specify the features of copy-on-write. What hardware support is required to implement it? [CO4,K2]
9. Are non-volatile memory devices faster? Justify your answer. [CO5,K2]
10. How is linked list used to manage the free space? [CO5,K2]

Part – B ($5 \times 16 = 80$ marks)

11. a. i) With neat diagram elaborate the computer system architecture. (10) [CO1,K1]
- ii) Outline the differences between symmetric and asymmetric multiprocessing. What are the advantages and disadvantages of multiprocessor systems? (6) [CO1,K2]

(OR)

- b. i) What is system calls? Outline the different types of system calls supported by an operating system. (10) [CO1,K1]
- ii) Specify the main advantage and disadvantage of the microkernel approach in the system design. How do user programs and system services interact in a micro kernel architecture? (6) [CO1,K2]

12. a. Consider the following set of processes with the length of the CPU burst give in (16) [CO2,K3] milliseconds.

Process	Burst Time	Priority
A	2	2
B	5	1
C	6	4
D	8	2
E	4	3

Draw the Gantt charts and calculate the average turnaround time and average waiting time of these processes using the FCFS, SJF, priority and RR (quantum = 3) scheduling algorithms.

(OR)

- b. Consider two cooperating processes are executing in the system. The processes (16) [CO2,K3] want to communicate each other to exchange data and information. Propose any two communication models to enable information sharing between the processes with necessary example.

13. a. Consider the following snapshot of a system. (16) [CO3,K3]

	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	2	0	0	1	4	2	1	2	3	3	2	1
P1	3	1	2	1	5	2	5	2				
P2	2	1	0	3	2	3	1	6				
P3	1	3	1	2	1	4	2	4				
P4	1	4	3	2	3	6	6	5				

Using banker's algorithm illustrate that the system is in a safe state by demonstrating an order in which the process may complete.

(OR)

- b. i) Propose a dead lock-free solution to the dining-philosophers problem using (8) [CO3,K3] monitors.
 ii) Provide a solution to address the readers-writers synchronization problem. (8) [CO3,K3]

14. a. i) Given six memory partitions of 300 KB, 600 KB, 400 KB, 250 KB, 700 KB (8) [CO4,K3] and 135 KB (in order), how would the first-fit, best-fit and worst-fit algorithms place processes of size 116 KB, 500 KB, 359 KB, 200 KB and 377 KB? Rank the algorithms in terms of how efficiently they use memory.

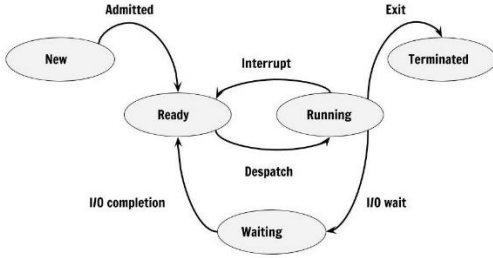
- ii) Elaborate any two types of page tables with neat diagram. (8) [CO4,K2]

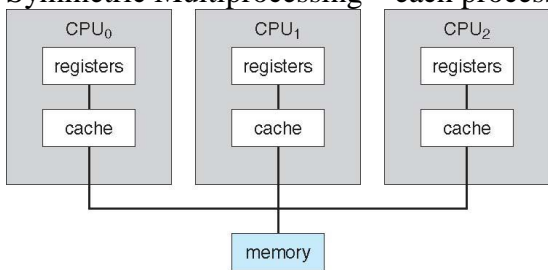
(OR)

- b. i) Consider the following page reference string 7, 2, 1, 3, 2, 5, 4, 3, 6, 7, 7, 1, 5, 0, 4, 6, 2, 3, 0, 1 Assuming demand paging with four frames, how many page faults would occur for the LRU, FIFO and optimal page replacement algorithms? (8) [CO4,K3]
- ii) What is thrashing? Discuss the cause of thrashing? How does the system detect thrashing? Once if detects thrashing what can the system do no eliminate this problem? (8) [CO4,K2]
15. a. Suppose that a disk drive has 6000 cylinders numbered 0 to 5999. The drive is currently serving a request at cylinder 2250 and the previous request was at cylinder 1950. The queue of pending requests in FIFO order 2065, 1212, 3296, 2500, 644, 1618, 456, 4523, 5600, 3681. Starting from the current head position what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending request for FCFS, SSTF, SCAN, LOOK and C-SCAN disk scheduling algorithms? (16) [CO5,K3]
- (OR)
- b. The users of the computer system usually store many files on the same disk. The main problem is how to allocate space to these files so that disk space is utilized effectively and files can be accessed quickly. Propose the methods to effectively allocate the disk space and elaborate how does it solves space utilization problem. (16) [CO5,K3]

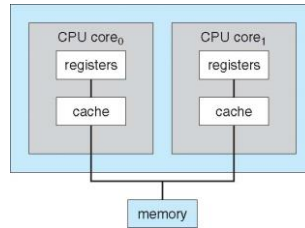
Bloom's Taxonomy Level	Remembering (K1)	Understanding (K2)	Applying (K3)	Analysing (K4)	Evaluating (K5)	Creating (K6)
Percentage	17	21	62	—	—	—

BTech Degree Exam May 2023
Fifth Semester
Information Technology
20ITT52 Operating System (R-2020)

PART - A		
1.	<p>The operating system (OS) plays a critical role in a computer system. Here are some of its key functions: (Any 2)</p> <p style="text-align: center;">Resource Management, Process Management, Memory Management, File System Management, Device Management, User Interface, Security and Protection, Error Handling, Networking</p>	(2)
2.	<p>Linker: The linker is a program that combines multiple object files, generated from source code compilation, into a single executable file or a library. Its main purpose is to resolve symbols and references between different object files and generate a cohesive executable program.</p> <p>Loader: The loader, sometimes called the runtime linker, is responsible for loading an executable program into memory and preparing it for execution. It is typically part of the operating system.</p>	(2)
3.	<p style="text-align: center;">Process State Diagram</p>  <pre> graph LR New([New]) -- Admitted --> Ready([Ready]) Ready -- Despatch --> Running([Running]) Running -- Interrupt --> Ready Running -- I/O wait --> Waiting([Waiting]) Waiting -- I/O completion --> Ready Running -- Exit --> Terminated([Terminated]) </pre>	(2)
4.	<p>Preemptive Scheduling is a type of CPU scheduling in which the resources (CPU Cycle) have been allocated to a process for a limited amount of time. In this type of scheduling, a process can be interrupted when it is being executed.</p> <p>Non-Preemptive Scheduling is one in which once the resources (CPU Cycle) have been allocated to a process, the process holds it until it completes its burst time or switches to the 'wait' state.</p>	(2)
5.	<p>Spin Lock:</p> <p>It is a locking mechanism. It enables a thread to wait for the lock to become ready, i.e., the thread can wait in a loop or spin until the lock is ready. It is only held for a short time, and it is useful in a multiprocessor system. The thread holds the spinlock until it is released after acquiring the lock. In some implementations, the spinlock is automatically released if the thread holding the lock is blocked or goes to sleep state.</p>	(2)
6.	<p>Deadlock Prevention</p> <p>Mutual Exclusion, Hold and wait, No Preemption, Circular Wait, Detection and Recovery</p>	(2)
7.	<p>Internal Fragmentation</p> <p>Internal fragmentation is defined as the difference between memory allocated and the memory space required by a process. Internal fragmentation occurs when the size of a process is larger than the memory required. In internal fragmentation, a memory block assigned to a process is bigger. Some portion of the memory is left unused, as it cannot be used by another process.</p>	(2)

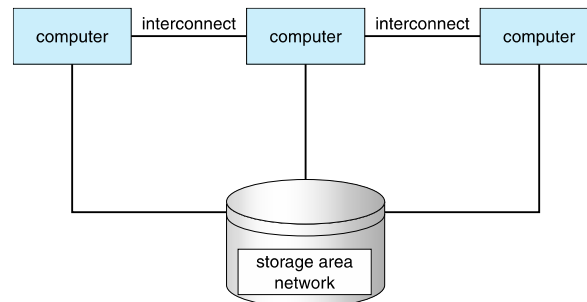
	<p>External Fragmentation</p> <p>External fragmentation is the unused space that is left between the fragments of non-contiguous memory. These unused spaces are too small to help a new process. The total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.</p>	
8.	<p>Copy on Write allows processes to share pages rather than each having a separate copy of the pages. However, when one process tried to write to a shared page, then a trap is generated and the OS makes a separate copy of the page for each process. This is commonly used in a fork() operation where the child is supposed to have a complete copy of the parent address space. Rather than create a separate copy, the OS allows the parent and child to share the parent's pages. However, since each is supposed to have its own private copy of the pages, the pages are copied when one of them attempts a write.</p> <p>The hardware support required to implement is simply the following: on each memory access, the page table needs to be consulted to check whether the page is write-protected. If it is indeed write-protected, a trap would occur and the operating system could resolve the issue.</p>	(2)
9.	<p>Non-volatile memory refers to a type of computer memory that retains data even when the power is turned off. Examples of non-volatile memory include solid-state drives (SSDs), flash memory, and magnetic hard drives. It is important to note that non-volatile memory is typically slower than volatile memory (e.g., RAM) in terms of access times. Here's an explanation of why non-volatile memory is slower:</p>	(2)
10.	<p>A linked list is another approach for free space management in an operating system. In it, all the free blocks inside a disk are linked together in a linked list. These free blocks on the disk are linked together by a pointer. These pointers of the free block contain the address of the next free block and the last pointer of the list points to null which indicates the end of the linked list.</p>	(2)
Part - B		
11.a.i	<p>Computer System Architecture:</p> <ul style="list-style-type: none"> • Most systems use a single general-purpose processor <ul style="list-style-type: none"> ○ Most systems have special-purpose processors as well • Multiprocessors systems growing in use and importance <ul style="list-style-type: none"> ○ Also known as parallel systems, tightly-coupled systems ○ Advantages include: <ul style="list-style-type: none"> ▪ Increased throughput ▪ Economy of scale ▪ Increased reliability – graceful degradation or fault tolerance ○ Two types: <ul style="list-style-type: none"> ▪ Asymmetric Multiprocessing – each processor is assigned a specie task. ▪ Symmetric Multiprocessing – each processor performs all tasks  <p>Dual core:</p>	(10)

- Multi-chip and multicore
- Systems containing all chips
 - Chassis containing multiple separate systems



Cluster Systems:

- Like multiprocessor systems, but multiple systems working together
 - Usually sharing storage via a storage-area network (SAN)
 - Provides a high-availability service which survives failures
 - Asymmetric clustering has one machine in hot-standby mode
 - Symmetric clustering has multiple nodes running applications, monitoring each other
 - Some clusters are for high-performance computing (HPC)
 - Applications must be written to use parallelization
 - Some have distributed lock manager (DLM) to avoid conflicting operations



11.a.ii Symmetric Multiprocessing (SMP) and Asymmetric Multiprocessing (AMP) are two different approaches to multiprocessing, which involve the simultaneous execution of multiple tasks or processes on a computer system. Here are the key differences between SMP and AMP:

Symmetric Multiprocessing (SMP):

1. Definition: In SMP, all processors are considered equal and have the same access to the system's resources. Each processor can perform any task or execute any process.
2. Load Balancing: SMP systems distribute the workload evenly across all processors. Tasks can be dynamically assigned to any available processor, ensuring efficient resource utilization.
3. Communication: SMP processors typically share a common memory space, allowing them to communicate and share data easily.
4. Scalability: SMP systems can be easily scaled by adding more processors to the system, enabling increased computing power and performance.
5. Complexity: SMP systems require sophisticated hardware and software designs to ensure proper coordination and synchronization among the processors.
6. Programming Model: SMP systems utilize a shared-memory programming model, where multiple threads or processes can access shared data directly.

	<p>Asymmetric Multiprocessing (AMP):</p> <ol style="list-style-type: none"> 1. Definition: In AMP, each processor is assigned a specific role or task. There is typically a master processor responsible for managing the system and other processors dedicated to executing specific functions. 2. Load Balancing: AMP systems distribute different tasks or processes to different processors based on their predefined roles. The workload is not evenly balanced across processors. 3. Communication: Communication between processors in an AMP system may require explicit coordination mechanisms since processors may have separate memory spaces and limited interprocessor communication. 4. Scalability: Adding more processors to an AMP system can be challenging as it may involve redefining the roles and responsibilities of each processor. 5. Simplified Design: Compared to SMP, AMP systems have a simpler design and require less complex hardware and software coordination. 6. Programming Model: AMP systems often use a message-passing programming model, where processors communicate by passing messages rather than directly accessing shared memory. <p>Advantages of Multiprocessor Systems:</p> <ol style="list-style-type: none"> 1. Increased Performance: Multiprocessor systems can handle multiple tasks simultaneously, leading to improved performance and faster execution of programs. 2. Enhanced Reliability: Multiprocessor systems can provide redundancy and fault tolerance. If one processor fails, the system can continue running on the remaining processors. 3. Better Resource Utilization: Multiprocessor systems allow for better utilization of system resources by distributing the workload across multiple processors. 4. Scalability: Multiprocessor systems can be easily scaled by adding more processors, providing a scalable solution to meet increasing computational demands. <p>Disadvantages of Multiprocessor Systems:</p> <ol style="list-style-type: none"> 1. Complexity: Multiprocessor systems are more complex to design, implement, and manage than single-processor systems. 2. Software Challenges: Developing software for multiprocessor systems can be challenging due to issues such as thread synchronization, shared memory access, and load balancing. 3. Cost: Multiprocessor systems tend to be more expensive than single-processor systems due to the additional hardware and coordination requirements. 4. Limited Scalability: Scaling certain multiprocessor systems beyond a certain point can become difficult due to constraints such as communication overhead and synchronization delays. 	
11.b.i	<p>System calls are the interface between user-level applications and the operating system. They are functions provided by the operating system that allow programs to request services from the kernel or access privileged operations that are otherwise restricted. System calls provide a way for user programs to interact with the underlying operating system and utilize its functionalities.</p>	(10)

	<p>Different operating systems may support different sets of system calls based on their design and intended usage. However, here are some common types of system calls supported by many operating systems:</p> <ol style="list-style-type: none"> 1. Process Control System Calls: <ul style="list-style-type: none"> ○ Fork: Creates a new process by duplicating the existing process. ○ Exec: Loads a new program into the current process's memory space. ○ Wait: Suspends the execution of a process until one of its child processes exits. ○ Exit: Terminates the execution of the current process and returns resources to the system. 2. File Management System Calls: <ul style="list-style-type: none"> ○ Open: Opens a file or creates a new file. ○ Close: Closes a file. ○ Read: Reads data from a file. ○ Write: Writes data to a file. ○ Seek: Changes the current position within a file. 3. Device Management System Calls: <ul style="list-style-type: none"> ○ Open: Opens a device for use. ○ Close: Closes a device. ○ Read: Reads data from a device. ○ Write: Writes data to a device. ○ Control: Sends control commands to a device. 4. Communication System Calls: <ul style="list-style-type: none"> ○ Socket: Creates a communication endpoint for network communication. ○ Bind: Associates a local address with a socket. ○ Listen: Puts a socket into a passive listening mode to accept incoming connections. ○ Accept: Accepts a connection on a socket. ○ Connect: Establishes a connection to a remote socket. 5. Memory Management System Calls: <ul style="list-style-type: none"> ○ Allocate: Requests memory allocation from the operating system. ○ Free: Releases allocated memory back to the operating system. 6. File System System Calls: <ul style="list-style-type: none"> ○ Mount: Attaches a file system to the directory tree. ○ Unmount: Detaches a file system from the directory tree. ○ Create: Creates a new file. ○ Delete: Deletes a file. ○ Rename: Renames a file. ○ List: Lists files in a directory. 7. Process Synchronization System Calls: <ul style="list-style-type: none"> ○ Lock: Acquires a lock to provide mutual exclusion. ○ Unlock: Releases a previously acquired lock. ○ Wait: Suspends the execution of a process until a certain condition is met. ○ Signal: Sends a signal to a process to notify or interrupt it. 	
11.b.ii	<p>The microkernel approach is a system design principle that advocates for a minimalistic kernel with a small set of essential functionalities, while moving most operating system services and functionalities to user-space processes. Here are the main advantage and disadvantage of the microkernel approach:</p>	(6)

Main Advantage of the Microkernel Approach:

1. **Modularity and Extensibility:** The microkernel design promotes modularity by keeping the kernel small and focusing on essential services, such as process management and interprocess communication. Additional services, such as file systems, device drivers, and networking, are implemented as user-space processes or servers. This modularity allows for easier extensibility and customization of the operating system by adding or replacing specific components without modifying the kernel.

Main Disadvantage of the Microkernel Approach:

1. **Performance Overhead:** The microkernel approach involves more interprocess communication and message passing between user-space processes for system services. This can introduce higher overhead compared to a monolithic kernel design, where services are tightly integrated within the kernel. The additional context switches and data transfers between user-space and kernel-space can impact performance, particularly for latency-sensitive operations.

In a microkernel architecture, user programs and system services interact through interprocess communication (IPC) mechanisms provided by the microkernel. Here's a simplified overview of the interaction:

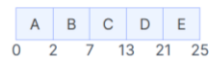
1. **User Programs:** User programs execute in user-space and make use of the services provided by the operating system. These programs interact with the system services through well-defined interfaces or APIs.
2. **System Services:** System services, such as file systems, device drivers, and networking protocols, are implemented as separate user-space processes or servers. They run independently of the kernel and communicate with user programs and other services using IPC mechanisms provided by the microkernel.
3. **Microkernel:** The microkernel provides a minimal set of core functionalities, such as process management, memory management, and interprocess communication. It handles basic operations and acts as a mediator between user programs and system services. It provides IPC mechanisms, such as message passing or shared memory, for communication between user programs and services.
4. **Interprocess Communication:** User programs can request services from the system by sending messages or making function calls to the appropriate service processes. The microkernel facilitates the communication between the user programs and the corresponding service processes. This communication can involve passing data, making requests, or receiving responses.

12.a

FCFS:

(16)

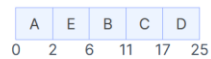
Gantt Chart



Job	Arrival Time	Burst Time	Finish Time	Turnaround Time	Waiting Time
A	0	2	2	2	0
B	0	5	7	7	2
C	0	6	13	13	7
D	0	8	21	21	13
E	0	4	25	25	21
Average				$68 / 5 = 13.6$	$43 / 5 = 8.6$

SJF

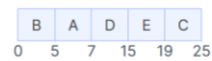
Gantt Chart



Job	Arrival Time	Burst Time	Finish Time	Turnaround Time	Waiting Time
A	0	2	2	2	0
B	0	5	11	11	6
C	0	6	17	17	11
D	0	8	25	25	17
E	0	4	6	6	2
Average				$61 / 5 = 12.2$	$36 / 5 = 7.2$

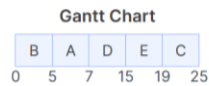
Priority:

Gantt Chart



Job	Arrival Time	Burst Time	Finish Time	Turnaround Time	Waiting Time
A	0	2	7	7	5
B	0	5	5	5	0
C	0	6	25	25	19
D	0	8	15	15	7
E	0	4	19	19	15
Average				$71 / 5 = 14.2$	$46 / 5 = 9.2$

RR



Job	Arrival Time	Burst Time	Finish Time	Turnaround Time	Waiting Time
A	0	2	7	7	5
B	0	5	5	5	0
C	0	6	25	25	19
D	0	8	15	15	7
E	0	4	19	19	15
Average				$71 / 5 = 14.2$	$46 / 5 = 9.2$

12.b Processes can communicate with each other through following methods: (Any two)
Shared Memory Method

(16)

There are two processes: Producer and Consumer. The producer produces some items and the Consumer consumes that item. The two processes share a common space or memory location known as a buffer where the item produced by the Producer is stored and from which the Consumer consumes the item if needed. There are two versions of this problem: the first one is known as the unbounded buffer problem in which the Producer can keep on producing items and there is no limit on the size of the buffer, the second one is known as the bounded buffer problem in which the Producer can produce up to a certain number of items before it starts waiting for Consumer to consume it. We will discuss the bounded buffer problem. First, the Producer and the Consumer will share some common memory, then the producer will start producing items. If the total produced item is equal to the size of the buffer, the producer will wait to get it consumed by the Consumer. Similarly, the consumer will first check for the availability of the item. If no item is available, the Consumer will wait for the Producer to produce it. If there are items available, Consumer will consume them. The pseudo-code to demonstrate is provided below

Message passing

- Mechanism for processes to communicate and to synchronize their actions
- Message system – processes communicate with each other without resorting to shared variables
- IPC facility provides two operations:
 - **send**(message)
 - **receive**(message)
- The *message* size is either fixed or variable
- If processes *P* and *Q* wish to communicate, they need to:
- Establish a **communication link** between them
- Exchange messages via send/receive

Direct communication

- Processes must name each other explicitly: (Symmetry)
 - **send** (*P*, *message*) – send a message to process *P*
 - **receive**(*Q*, *message*) – receive a message from process *Q*
- Properties of communication link
 - Links are established automatically

- A link is associated with exactly one pair of communicating processes
- Between each pair there exists exactly one link
- The link may be unidirectional, but is usually bi-directional
- Sender names the recipient (Asymmetry)
 - **send** (*P, message*) – send a message to process P
 - **receive**(*id, message*)- receive the message based on id

Indirect communication

- Messages are directed and received from mailboxes (also referred to as ports)
 - Each mailbox has a unique id
 - Processes can communicate only if they share a mailbox
- Properties of communication link
 - Link established only if processes share a common mailbox
 - A link may be associated with many processes
 - Each pair of processes may share several communication links
 - Link may be unidirectional or bi-directional
- Operations
 - create a new mailbox (port)
 - send and receive messages through mailbox
 - destroy a mailbox
- Primitives are defined as:

send(*A, message*) – send a message to mailbox A

receive(*A, message*) – receive a message from mailbox A

13.a

Need Matrix

(16)

Need Matrix:

2	2	1	1
2	1	3	1
0	2	1	3
0	1	1	2
2	2	3	3

Process List:

P0
P1
P2
P3
P4

Allocation Matrix:

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

Maximum Matrix:

4	2	1	2
5	2	5	2
2	3	1	6
1	4	2	4
3	6	6	5

Available Resources:

3	3	2	1
5	3	2	2
6	6	3	4
7	10	6	6
10	11	8	7
12	12	8	10

Need Matrix:

2	2	1	1
2	1	3	1
0	2	1	3
0	1	1	2
2	2	3	3

Safe Sequence is - P0 -> P3 -> P4 -> P1 -> P2

To illustrate monitor concepts by presenting a deadlock-free solution to the dining-philosophers problem. Monitor is used to control access to state variables and condition variables. It only tells when to enter and exit the segment. This solution imposes the restriction that a philosopher may pick up her chopsticks only if both of them are available. To code this solution, we need to distinguish among three states in which we may find a philosopher. For this purpose, we introduce the following data structure: **THINKING** – When philosopher doesn't want to gain access to either fork. **HUNGRY** – When philosopher wants to enter the critical section. **EATING** – When philosopher has got both the forks, i.e., he has entered the section. Philosopher i can set the variable $state[i] = EATING$ only if her two neighbors are not eating ($state[(i+4) \% 5] \neq EATING$) and ($state[(i+1) \% 5] \neq EATING$).

monitor DP

```
{
    status state[5];
    condition self[5];

    // Pickup chopsticks
    Pickup(int i)
    {
        // indicate that I'm hungry
        state[i] = hungry;

        // set state to eating in test()
        // only if my left and right neighbors
        // are not eating
        test(i);

        // if unable to eat, wait to be signaled
        if (state[i] != eating)
            self[i].wait;
    }

    // Put down chopsticks
    Putdown(int i)
    {

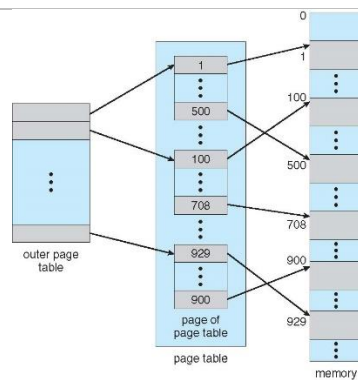
        // indicate that I'm thinking
        state[i] = thinking;

        // if right neighbor R=(i+1)%5 is hungry and
        // both of R's neighbors are not eating,
        // set R's state to eating and wake it up by
        // signaling R's CV
        test((i + 1) % 5);
        test((i + 4) % 5);
    }

    test(int i)
    {
```

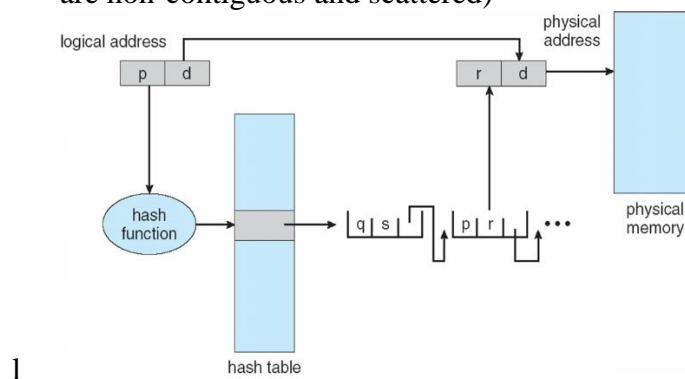
	<pre> if (state[(i + 1) % 5] != eating && state[(i + 4) % 5] != eating && state[i] == hungry) { // indicate that I'm eating state[i] = eating; // signal() has no effect during Pickup(), // but is important to wake up waiting // hungry philosophers during Putdown() self[i].signal(); } } init() { // Execution of Pickup(), Putdown() and test() // are all mutually exclusive, // i.e. only one at a time can be executing for i = 0 to 4 // Verify that this monitor-based solution is // deadlock free and mutually exclusive in that // no 2 neighbors can eat simultaneously state[i] = thinking; } } // end of monitor </pre>	
13.b.ii	<p>Readers Writers Problem:</p> <ul style="list-style-type: none"> • A data set is shared among a number of concurrent processes <ul style="list-style-type: none"> ○ Readers – only read the data set; they do not perform any updates ○ Writers – can both read and write • Problem – allow multiple readers to read at the same time <ul style="list-style-type: none"> ○ Only one single writer can access the shared data at the same time • Several variations of how readers and writers are considered – all involve some form of priorities • Shared Data <ul style="list-style-type: none"> ○ Data set ○ Semaphore rw_mutex initialized to 1 ○ Semaphore mutex initialized to 1 ○ Integer read_count initialized to 0 <p>The structure of a writer process</p> <pre> do { wait(rw_mutex); ... /* writing is performed */ ... </pre>	(8)

	<pre>signal(rw_mutex); } while (true); The structure of a reader process do { wait(mutex); read_count++; if (read_count == 1) wait(rw_mutex); signal(mutex); ... /* reading is performed */ ... wait(mutex); read count--; if (read_count == 0) signal(rw_mutex); signal(mutex); } while (true);</pre>																																					
14.a.i	<p>First Fit</p> <table border="1"><tr><td>300</td><td>116</td></tr><tr><td>600</td><td>500</td></tr><tr><td>400</td><td>359</td></tr><tr><td>250</td><td>200</td></tr><tr><td>700</td><td>377</td></tr><tr><td>135</td><td></td></tr></table> <p>Best Fit</p> <table border="1"><tr><td>300</td><td></td></tr><tr><td>600</td><td>500</td></tr><tr><td>400</td><td>359</td></tr><tr><td>250</td><td>200</td></tr><tr><td>700</td><td>377</td></tr><tr><td>135</td><td>116</td></tr></table> <p>Worst Fit - 377 Not allocated</p> <table border="1"><tr><td>300</td><td>200</td></tr><tr><td>600</td><td>500</td></tr><tr><td>400</td><td>359</td></tr><tr><td>250</td><td></td></tr><tr><td>700</td><td>116</td></tr><tr><td>135</td><td></td></tr></table>	300	116	600	500	400	359	250	200	700	377	135		300		600	500	400	359	250	200	700	377	135	116	300	200	600	500	400	359	250		700	116	135		(8)
300	116																																					
600	500																																					
400	359																																					
250	200																																					
700	377																																					
135																																						
300																																						
600	500																																					
400	359																																					
250	200																																					
700	377																																					
135	116																																					
300	200																																					
600	500																																					
400	359																																					
250																																						
700	116																																					
135																																						
14.a.ii	<p>Types of Page table (Any two)</p> <p>Hierarchical Paging</p> <ul style="list-style-type: none">• Break up the logical address space into multiple page tables• A simple technique is a two-level page table• We then page the page table	(8)																																				



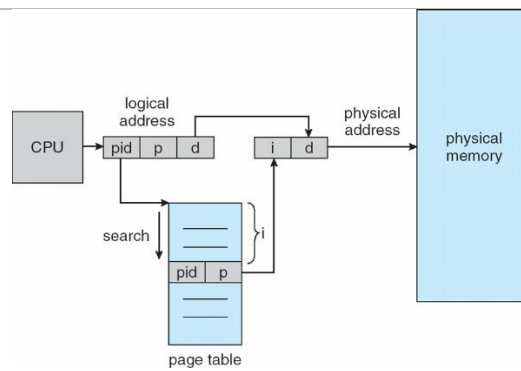
Hashed Page Tables

- n Common in address spaces > 32 bits
- n The virtual page number is hashed into a page table
 - 1 This page table contains a chain of elements hashing to the same location
- n Each element contains (1) the virtual page number (2) the value of the mapped page frame (3) a pointer to the next element
- n Virtual page numbers are compared in this chain searching for a match
 - 1 If a match is found, the corresponding physical frame is extracted
- n Variation for 64-bit addresses is **clustered page tables**
 - 1 Similar to hashed but each entry refers to several pages (such as 16) rather than 1
 - 1 Especially useful for **sparse** address spaces (where memory references are non-contiguous and scattered)



Inverted Page Tables

- n Rather than each process having a page table and keeping track of all possible logical pages, track all physical pages
- n One entry for each real page of memory
- n Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page
- n Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs
- n Use hash table to limit the search to one — or at most a few — page-table entries
 - 1 TLB can accelerate access
- n But how to implement shared memory?
 - 1 One mapping of a virtual address to the shared physical address



14.b.i FIFO

(8)

t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
ref		7	2	1	3	2	5	4	3	6	7	7	1	5	0	4	6	2	3	0	1
f		7	2	1	3	3	5	4	4	6	7	7	1	5	0	4	6	2	3	0	1
f			7	2	1	1	3	5	5	4	6	6	7	1	5	0	4	6	2	3	0
f				7	2	2	1	3	3	5	4	4	6	7	1	5	0	4	6	2	3
f					7	7	2	1	1	3	5	5	4	6	7	1	5	0	4	6	2
hit		X	X	X	X	✓	X	X	✓	X	X	✓	X	X	X	X	X	X	X	X	X
v							7	2		1	3		5	4	6	7	1	5		4	6

Page Faults: 17

LRU

t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
ref		7	2	1	3	2	5	4	3	6	7	7	1	5	0	4	6	2	3	0	1
f		7	2	1	3	2	5	4	3	6	7	7	1	5	0	4	6	2	3	0	1
f			7	2	1	3	2	5	4	3	6	6	7	1	5	0	4	6	2	3	0
f				7	2	1	3	2	5	4	3	3	6	7	1	5	0	4	6	2	3
f					7	7	1	3	2	5	4	4	3	6	7	1	5	0	4	6	2
hit		X	X	X	X	✓	X	X	✓	X	X	✓	X	X	X	X	X	X	X	X	X
v							7	1		2	5		4	3	6	7	1	5		4	6

Page Faults: 17

Optimal

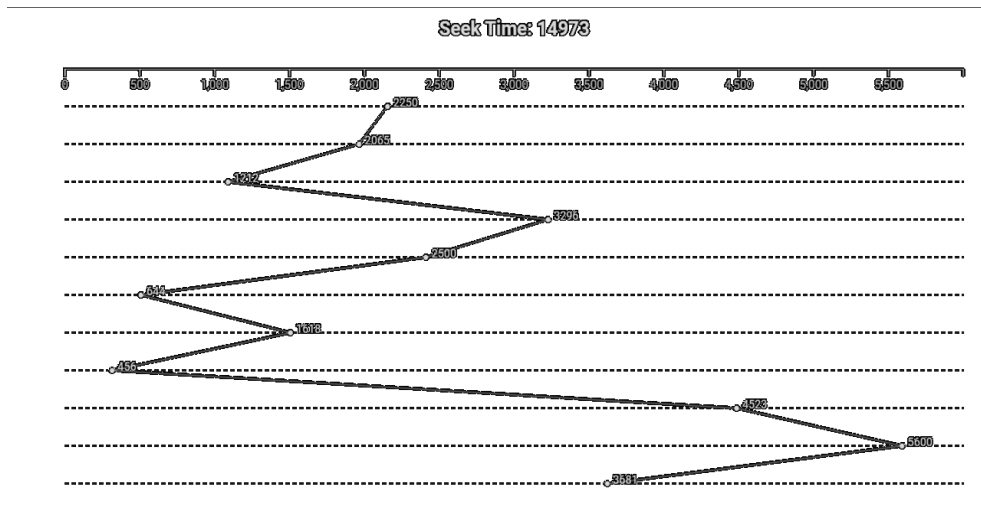
t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
ref		7	2	1	3	2	5	4	3	6	7	7	1	5	0	4	6	2	3	0	1
f		7	2	1	3	3	5	4	4	6	6	6	6	5	0	0	0	2	3	3	3
f			7	2	1	1	3	3	3	4	4	4	4	6	6	6	6	0	0	0	0
f				7	2	2	1	1	1	1	1	1	1	4	4	4	4	4	4	4	4
f					7	7	7	7	7	7	7	7	7	1	1	1	1	1	1	1	1

	<table><tr><td>hit</td><td></td><td>X</td><td>X</td><td>X</td><td>X</td><td>✓</td><td>X</td><td>X</td><td>✓</td><td>X</td><td>✓</td><td>✓</td><td>✓</td><td>X</td><td>X</td><td>✓</td><td>✓</td><td>X</td><td>X</td><td>✓</td><td>✓</td></tr><tr><td>v</td><td></td><td></td><td></td><td></td><td></td><td></td><td>2</td><td>5</td><td></td><td>3</td><td></td><td></td><td></td><td>7</td><td>5</td><td></td><td></td><td>6</td><td>2</td><td></td><td></td></tr></table>	hit		X	X	X	X	✓	X	X	✓	X	✓	✓	✓	X	X	✓	✓	X	X	✓	✓	v							2	5		3				7	5			6	2			
hit		X	X	X	X	✓	X	X	✓	X	✓	✓	✓	X	X	✓	✓	X	X	✓	✓																									
v							2	5		3				7	5			6	2																											
	Page faults:11																																													
14.b.ii	<p>Thrashing refers to a situation in computer systems, specifically in virtual memory management, where the system spends a significant amount of time and resources continuously swapping data between physical memory (RAM) and the disk, resulting in poor performance and low overall throughput.</p> <p>Thrashing typically occurs when a system is under heavy load and does not have enough physical memory to hold all the necessary data and processes. When the available physical memory becomes insufficient, the operating system starts swapping out pages of memory to disk to make room for new pages. However, if the system continues to experience a high demand for memory and the swapped-out pages are immediately needed again, they have to be swapped back in from the disk. This constant swapping in and out of pages creates a cycle of excessive disk I/O and consumes significant CPU and disk resources, leading to a degradation of system performance.</p> <p>Thrashing can be caused by several factors, including:</p> <ol style="list-style-type: none">1. Insufficient Physical Memory: When the system does not have enough RAM to accommodate all the active processes and their working sets, frequent swapping occurs.2. Improper Memory Allocation: If the system does not effectively allocate memory resources, it may allocate memory to processes in a way that results in excessive page faults and swapping.3. Overcommitment of Memory: If the system allows more processes to run than it can support with available physical memory, thrashing may occur as the system tries to keep up with the demand.4. Memory Fragmentation: Fragmented memory can make it difficult for the operating system to allocate contiguous blocks of memory, leading to increased page faults and swapping. <p>The system can detect thrashing by evaluating the level of CPU utilization as compared to the level of multiprogramming. It can be eliminated by reducing the level of multiprogramming.</p>	(8)																																												

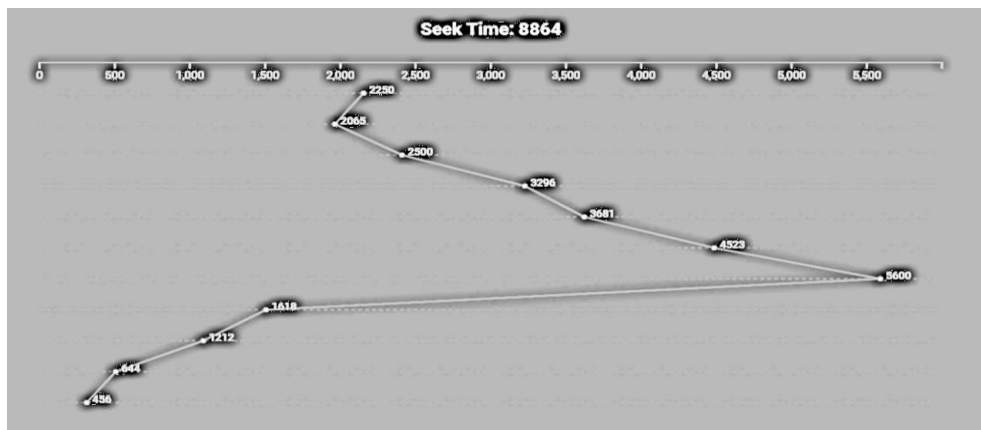
15.a

FCFS:

(16)

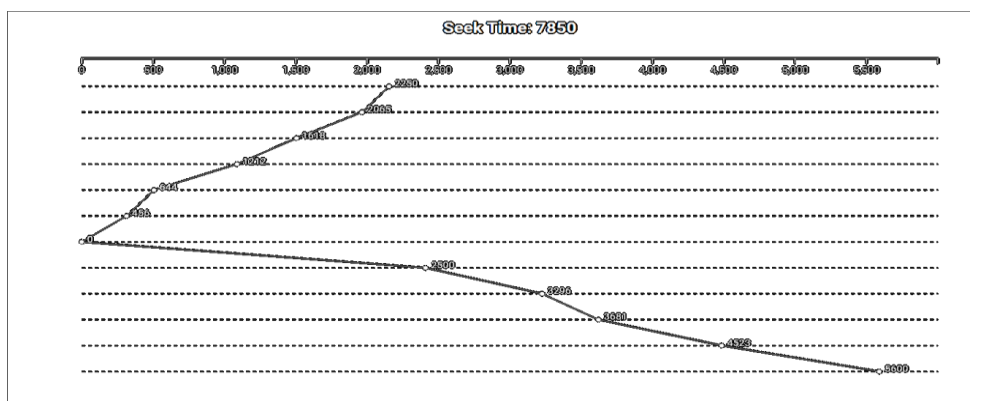


SSTF:

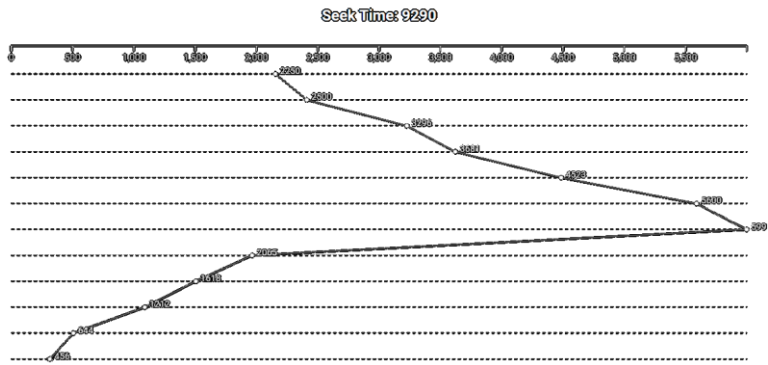


SCAN:

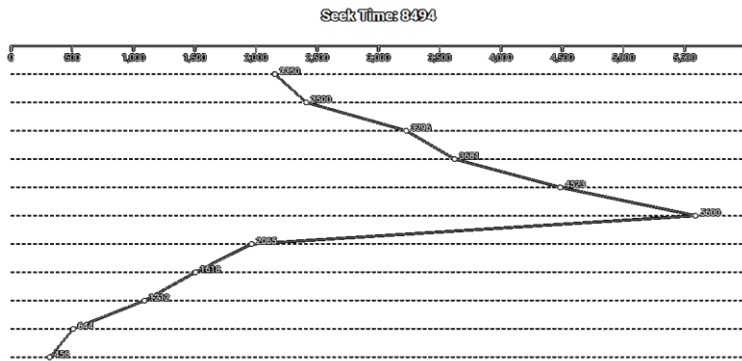
(left)



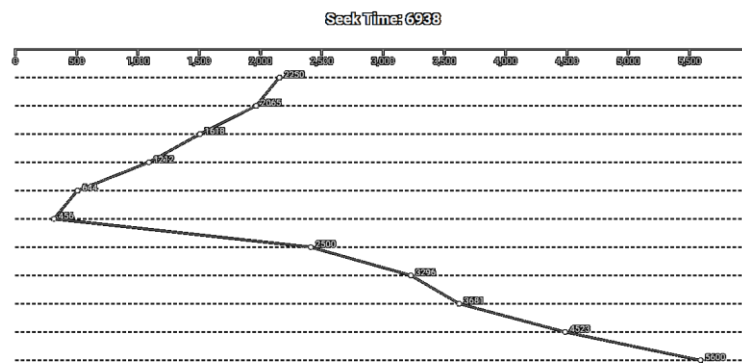
(OR)



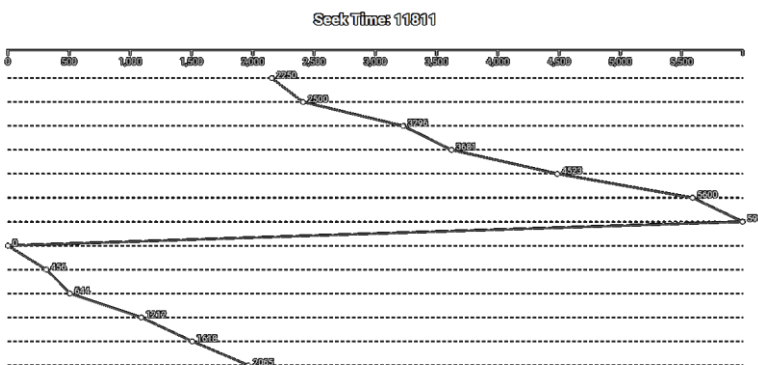
LOOK:



(OR)



C-SCAN:



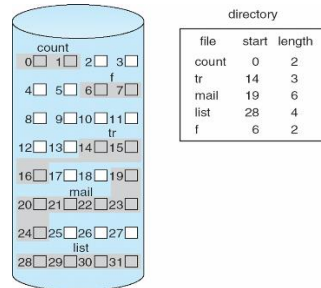
15.b.

To effectively allocate disk space for storing files in a computer system, there are several techniques and strategies that can be employed:

(16)

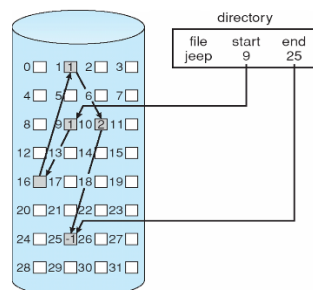
1. Contiguous Allocation:

- In contiguous allocation, each file is allocated a contiguous block of disk space.
- This method provides fast and efficient file access since the entire file is stored in a continuous block on the disk.
- However, it can lead to fragmentation, where free space becomes scattered, making it difficult to allocate contiguous blocks for larger files.



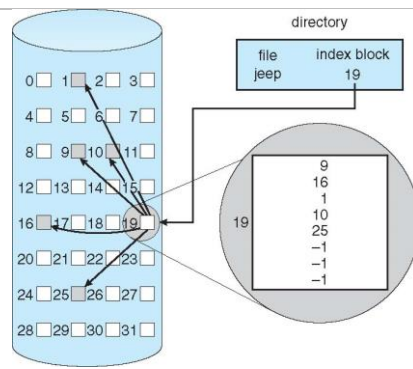
2. Linked Allocation:

- In linked allocation, each file is divided into blocks, and each block contains a pointer to the next block in the file.
- Free space is managed using a linked list, where each node represents a block of free space.
- This method avoids fragmentation since files can be scattered across the disk, utilizing any available free blocks.
- However, accessing a specific portion of a file requires traversing the linked list, which can result in slower access times.



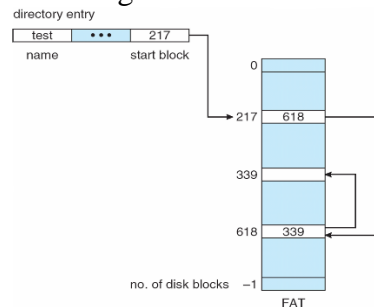
3. Indexed Allocation:

- In indexed allocation, each file has an associated index block that contains an index of all the disk blocks used by the file.
- The index block acts as a lookup table, allowing direct access to any block of the file.
- This method provides faster access compared to linked allocation but requires additional overhead for maintaining the index blocks.



4. File Allocation Table (FAT):

- FAT is a file system structure used in some operating systems, such as FAT16 and FAT32.
- It uses a file allocation table, which is a table that maps file clusters to disk blocks.
- The table contains entries for each cluster on the disk, indicating whether it is allocated to a file or free.
- This method provides efficient random access to files but can suffer from fragmentation over time.



Performance:

- Best method depends on file access type
 - Contiguous great for sequential and random
- Linked good for sequential, not random
- Declare access type at creation -> select either contiguous or linked
- Indexed more complex
 - Single block access could require 2 index block reads then data block read
 - Clustering can help improve throughput, reduce CPU overhead