CBCS SCHEME

USN | 1 | C | R | 2 | U | C | 2 | 1 | 3 | 3 |

BCS303

Third Semester B.E/B.Tech. Degree Examination, Dec.2025/Jan.2026
Operating Systems

Time: 3 hrs. Max. Marks:100

Note: *1. Answer any FIVE full questions, choosing ONE full question from each module.*
2. M : Marks , L: Bloom's level , C: Course outcomes.

		Module – 1	M	L	C
1	a.	Explain the various types of system calls with an example for each.	8	L2	CO1
	b.	Explain cloud computing its types and their services it offers.	8	L2	CO1
	c.	What is dual mode operation and what is the need of it?	4	L2	CO1
		OR			
2	a.	Explain the different architecture of OS starting from Simple Structure, Layered Structure, Micro Kernels, Modules and Hybrid Systems.	8	L4	CO1
	b.	Discuss the essential properties of the following types of systems : i. Time sharing system ii. Multi-programmed batch systems.	6 6	L2	CO1
		Module – 2			
3	a.	Explain Inter Process Communication.	6	L2	CO2
	b.	Discuss Multilevel Queue Scheduling Algorithm.	6	L4	CO2
	c.	Consider the set of 3 processes whose arrival time and burst time are given below.	8	L3	CO2

Process Id	Arrival Time	Burst Time
P1	0	9
P2	1	4
P3	2	9

If the CPU scheduling policy is SRTF, calculate the average waiting time and average turnaround time.

OR

4	a.	Compare User Level Threads and Kernel Level threads.	4	L4	CO2
	b.	Illustrate with a neat sketch, Process States and Process Control Block (PCB).	8	L2	CO2
	c.	Consider the set of 6 processes whose arrival time and burst time are given below.	8	L3	CO2

Process ID	Arrival Time	Burst Time
P_1	5	5
P_2	4	6
P_3	3	7
P_4	1	9
P_5	2	2
P_6	6	3

If the CPU scheduling policy is Round Robin with time quantum = 3, calculate Average Waiting time and Turnaround time.

Module – 3

5	a.	Discuss in detail the critical section problem and write the algorithm for producer consumer problem.	10	L2	CO3
	b.	Consider the following system using data structures in the Bankers Algorithm with resource type ABC Maximum instance present in the system A = 10, B = 5, C = 7.	10	L3	CO3

Process	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	3	3	2
P_1	2	0	0	3	2	2			
P_2	3	0	2	9	0	2			
P_3	2	1	1	2	2	2			
P_4	0	0	2	4	3	3			

i. Calculate Need Matrix
ii. Check whether system is safe or not.

OR

6	a.	Outline the solutions of Dinning – Philosopher problem.	5	L2	CO3
	b.	Describe a resource allocation graph with an example.	5	L4	CO3
	c	Using Bankers algorithm, solve the following problem :	10	L2	CO3

Process	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	0	1	2	0	0	1	2	1	5	2	0
P_1	1	0	0	0	1	7	5	0				
P_2	1	3	5	4	2	3	5	6				
P_3	0	6	3	2	0	6	5	2				
P_4	0	0	1	4	0	6	5	6				

i. Calculate the Need Matrix
ii. Check whether system is safe or not.

		Module – 4			
7	a.	Discuss the given memory management technique with diagram. i. Paging ii. Translation Look-Aside Buffer.	6 6	L2	CO4
	b.	Discuss about Contiguous Memory Allocation with a neat diagram.	8	L2	CO4
		OR			
8	a	Consider the reference string : 6, 1, 1, 2, 0, 3, 4, 6, 0, 2, 1, 2, 1, 2, 0, 3, 2, 1, 2, 0 For a memory with three frames and calculate number of page faults by using i. LRU replacement ii. FIFO replacement.	10	L3	CO4
	b.	Describe the process of demand paging in OS.	10	L2	CO4
		Module – 5			
9	a.	Explain in detail about directory and disc structure.	6	L2	CO5
	b.	Analyze the file system implementation.	6	L4	CO5
	c.	The requested tracks, in the order received are {176, 79, 34, 60, 92, 11, 41, 114} Apply the following disk scheduling algorithms starting track at 50. i) FCFS ii) SSTF. Calculate the total seek time.	8	L3	CO5
		OR			
10	a.	Explain Free Space Management with an example.	6	L2	CO2
	b.	Explain the Access Matrix method of system protection with the domain as objects and its implementation.	6	L2	CO2
	c.	The requested tracks, the order received are {176, 79, 34, 60, 92, 11, 41, 114} Apply the following disk scheduling algorithms starting track at 50. i) Look ii) C – Look Calculate the total seek time.	8	L3	CO3

* * * * *

MODULE 1

Q1a.
Solution:
Types of System Calls: ***(4M)***
Process Control
File Management
Device Management
Information maintenance
Communication
Protection

Example***: (4M)***
Create process, end, abort
load , execute
Create file, delete file
Read, write, reposition,
Get time or date client server
Control access

Q1b.
Solution:
Cloud computing
Delivers computing, storage, even apps as a service across a network
Types of cloud: Public cloud, private cloud, hybrid cloud ***(4M)***
Types of services: IaaS, Paas, Saas ***(4M)***

Q1c.
Solution:
An operating system is system software that acts as an intermediary between a user of a computer and the computer hardware. It is software that manages the computer hardware and allows the user to execute programs in a convenient and efficient manner.
Dual-Mode Operation:
The system can be assumed to work in two separate modes of operation:
1. User mode
2. Kernel mode (supervisor mode, system mode, or privileged mode).
A hardware bit of the computer, called the mode bit, is used to indicate the current mode: kernel (0) or user (1).
When the computer system is executing a user application, the system is in user mode. When a user application requests a service from the operating system (via a system call), the transition from user to kernel mode takes place. The dual mode of operation provides us with the means for protecting the operating system from errant users—and errant users from one another.
• The hardware allows privileged instructions to be executed only in kernel mode. If an attempt is made to execute a privileged instruction in user mode, the hardware does not execute the instruction but rather treats it as illegal and traps it to the operating system***. (4M)***

MODULE 1

Q2a.
Solution:
Simple Structure-MS-DOS
Layered approach-Networking Model (TCP/ UDP)
Microkernel systems-MACH
Modules-LINUX/ SOLARIS
Hybrid Systems (MAC-OS)
Explanation with diagram ***(8M)***

Q2b.
Solution:
Timesharing (Multitasking) ***(6M)***
Definition, Advantages and usage etc.
Multiprogramming (Batch Systems) ***(6M)***
Definition, Advantages and usage etc.

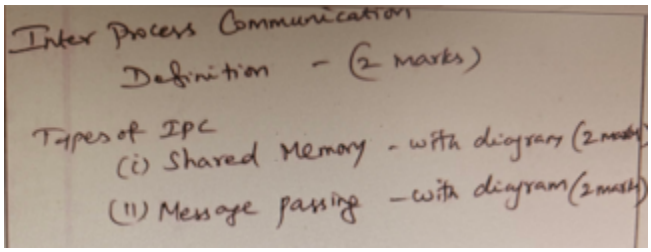
MODULE 2

Q3a.
Solution:
Interprocess Communication- Processes executing may be either co-operative or independent processes.
• Independent Processes – processes that cannot affect other processes or be affected by other processes executing in the system.
• Cooperating Processes – processes that can affect other processes or be affected by other processes executing in the system.
a) Direct communication: the sender and receiver must explicitly know each other's name. The syntax for send() and receive() functions are as follows-
• send (P, message) – send a message to process P
• receive(Q, message) – receive a message from process Q

Properties of communication link:
• A link is established automatically between every pair of processes that wants to communicate. The processes need to know only each other's identity to communicate.
• A link is associated with exactly one pair of communicating processes • Between each pair, there exists exactly one link.
b) Indirect communication uses shared mailboxes, or ports.
A mailbox or port is used to send and receive messages. Mailbox is an object into which messages can be sent and received. It has a unique ID. Using this identifier messages are sent and received.

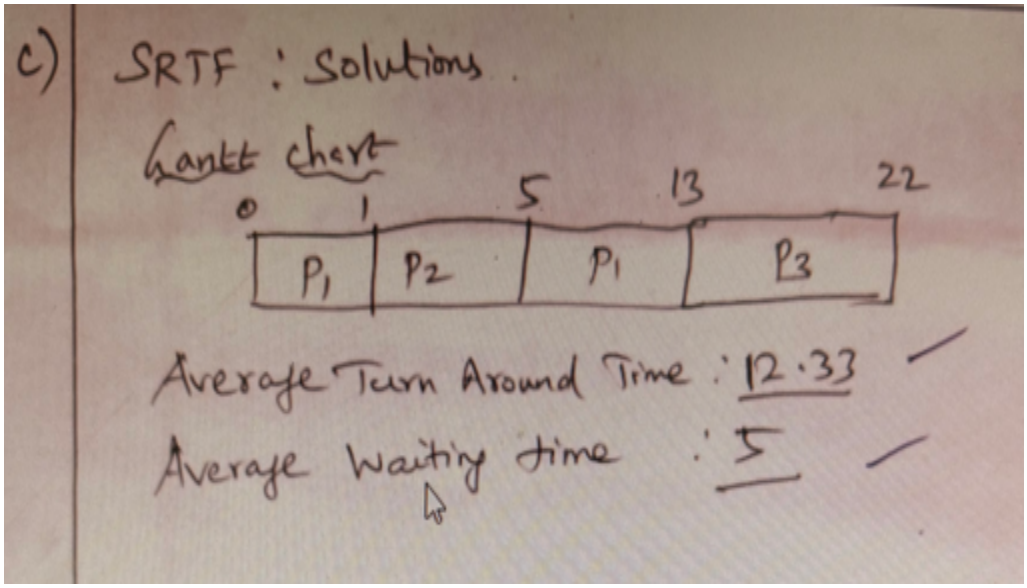
Two processes can communicate only if they have a shared mailbox. The send and receive functions are –
• send (A, message) – send a message to mailbox A
• receive (A, message) – receive a message from mailbox A

Properties of communication link:
• A link is established between a pair of processes only if they have a shared mailbox
• A link may be associated with more than two processes
• Between each pair of communicating processes, there may be any number of links, each link is associated with one mailbox.
• A mail box can be owned by the operating system. It must take steps to –
• create a new mailbox
• send and receive messages from mailbox
• delete mailboxes.



Q3b.
Solution:
Multilevel Queue Scheduling Algorithm:
Introduction ***(2M)***
Explanation with examples ***(3M)***
Diagram ***(1M)***

Q3c.
Solution:



MODULE 2

Q4a.
Solution:
Support for threads may be provided at either
1. The user level, for user threads or
2. By the kernel, for kernel threads.
• User-threads are supported above the kernel and are managed without kernel support. Kernel-threads are supported and managed directly by the OS.
• Three ways of establishing relationship between user-threads &kernel-threads:
1. Many-to-one model
2. One-to-one model and
3. Many-to-many model.
Many-to-One Model
• Many user-level threads are mapped to one kernel thread.
One-to-One Model
• Each user thread is mapped to a kernel thread.
Advantages:
▪ It provides more concurrency by allowing another thread to run when a thread makes a blocking system-call.
▪ Multiple threads can run in parallel on multiprocessors.
Many-to-Many Model
• Many user-level threads are multiplexed to a smaller number of kernel threads. ***(4M)***

Q4b.
Solution:
A process is a program under execution.
A Process has 5 states. Each process may be in one of the following states –***(4M)***
1. New - The process is in the stage of being created.
2. Ready - The process has all the resources it needs to run. It is waiting to be assigned to the processor.
3. Running – Instructions are being executed.
4. Waiting - The process is waiting for some event to occur. For example, the process may be waiting for keyboard input, disk access request, inter-process messages, a timer to go off, or a child process to finish.
5. Terminated - The process has completed its execution.

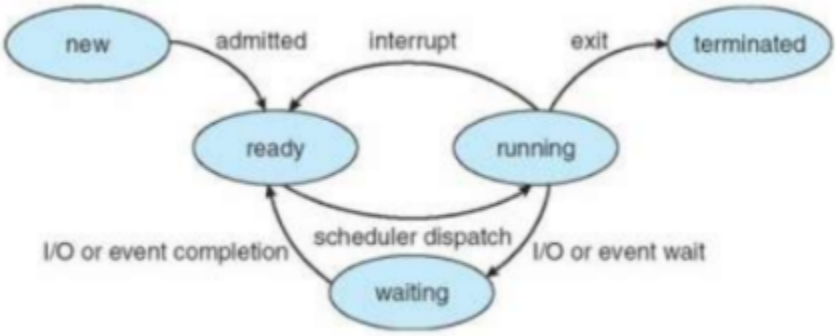


Figure: Diagram of process state

Process Control Block ***(4M)***
For each process there is a Process Control Block (PCB), which stores the process-specific information as shown below –
• Process State – The state of the process may be new, ready, running, waiting, and so on.
• Program counter – The counter indicates the address of the next instruction to be executed for this process.
• CPU registers - The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-purpose registers. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward.
• CPU scheduling information- This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.
• Memory-management information – This includes information such as the value of the base and limit registers, the page tables, or the segment tables.
• Accounting information – This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.
• I/O status information – This information includes the list of I/O devices allocated to the process, a list of open files, and so on.

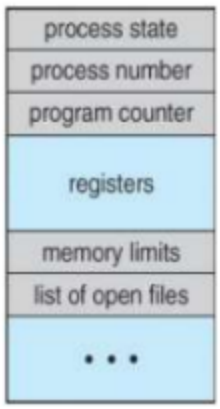
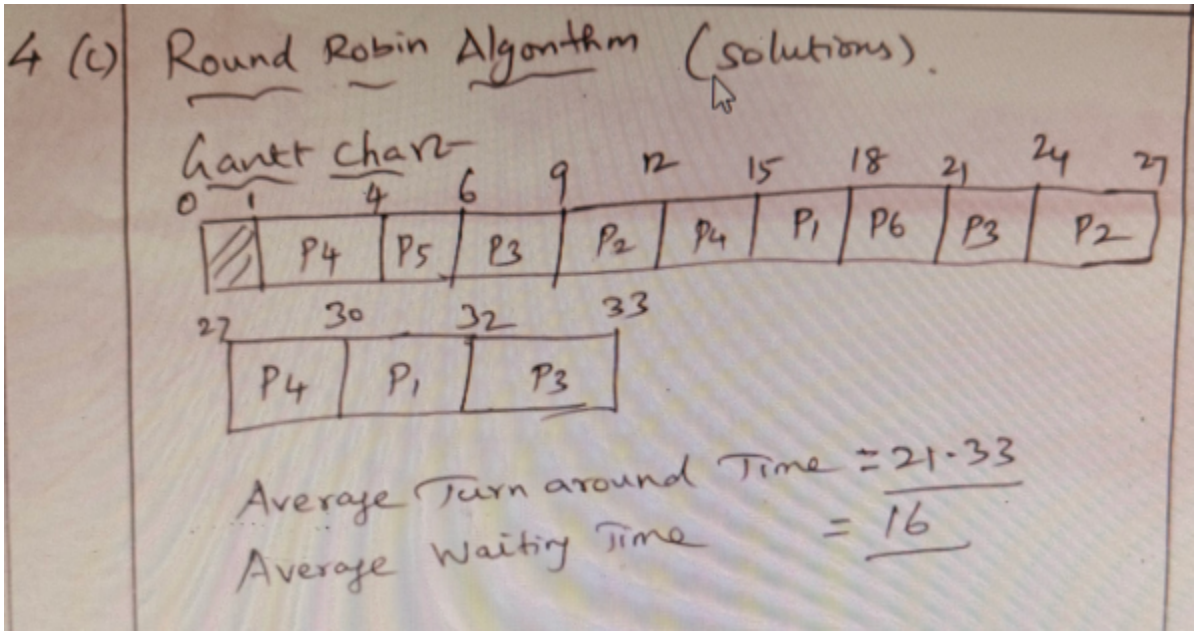


Figure: Process control block (PCB)

Q4c.
Solution:



(8M)

MODULE 3

Q5a.
Solution:
Critical Section (5M)
When one process is executing in its critical section, no other process is to be allowed to execute in its critical section. That is, no two processes are executing in their critical sections at the same time.
A solution to the critical-section problem must satisfy the following three requirements:
1. Mutual exclusion:If process Pi is executing in its critical section, then no other processes can be executing in their critical sections.
2. Progress:If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder sections can participate in deciding which will enter its critical section next, and this selection cannot be postponed indefinitely.
3. Bounded waiting:There exists a bound, or limit, 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.

Producer-Consumer Example Using Shared Memory ***(5M)***
• This is a classic example, in which one process is producing data and another process is consuming the data.
• The data is passed via an intermediary buffer (shared memory). The producer puts the data to the buffer and the consumer takes out the data from the buffer. A producer can produce one item while the consumer is consuming another item. The producer and consumer must be synchronized, so that the consumer does not try to consume an item that has not yet been produced. In this situation, the consumer must wait until an item is produced.
• There are two types of buffers into which information can be put –
• Unbounded buffer
• Bounded buffer

• With Unbounded buffer, there is no limit on the size of the buffer, and so on the data produced by producer. But the consumer may have to wait for new items.
• With bounded-buffer – As the buffer size is fixed. The producer has to wait if the buffer is full and the consumer has to wait if the buffer is empty.

- The producer process –

  Note that the buffer is full when [(in+1) % BUFFER_SIZE == out]

```
item nextProduced;

while (true) {
     /* produce an item in nextProduced */
     while ((in + 1) % BUFFER-SIZE) == out)
        ; /* do nothing */
     buffer[in] = nextProduced;
     in = (in + 1) % BUFFER_SIZE;
}
```

Figure The producer process.

- The consumer process –

  Note that the buffer is empty when [in == out]

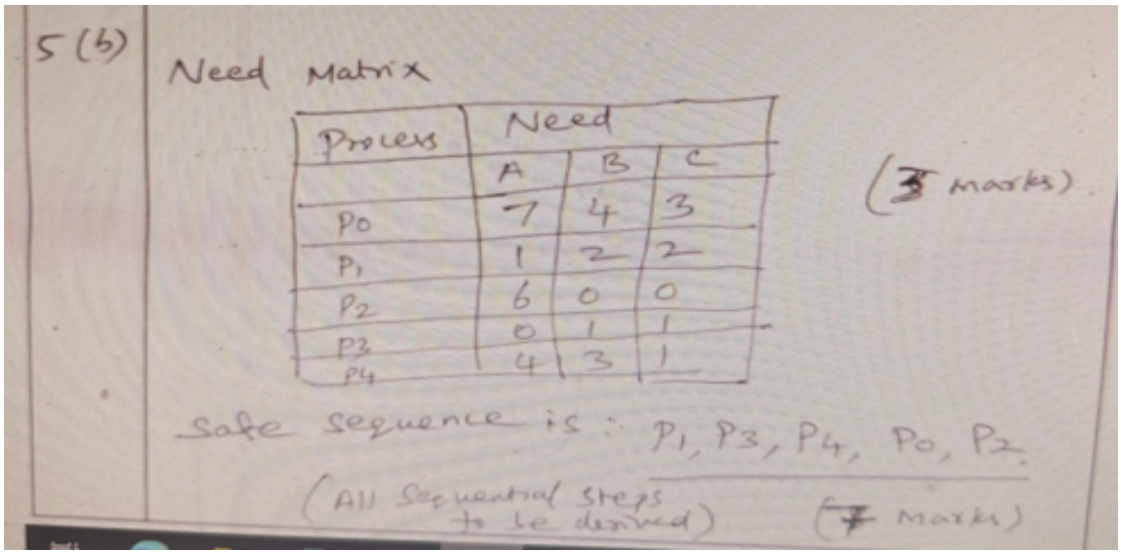
```
item nextConsumed;

while (true) {
     while (in == out)
        ; //do nothing

     nextConsumed = buffer[out];
     out = (out + 1) % BUFFER_SIZE;
     /* consume the item in nextConsumed */
}
```

Figure The consumer process.

Q5b.
Solution:


5 (b)

Need Matrix

Process	Need		
	A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

(3 marks)

Safe sequence is : P_1, P_3, P_4, P_0, P_2.

(All Sequential steps to be derived) (7 Marks)

Need Matrix: 3M
Safe Sequence: 7M

MODULE 3

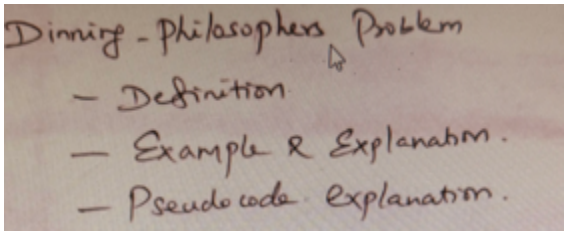
Q6a.
Solution:
Dining-Philosophers Problem ***(5M)***

Consider five philosophers who spend their lives thinking and eating. The philosophers share a circular table surrounded by five chairs, each belonging to one philosopher. In the center of the table is a bowl of rice, and the table is laid with five single chopsticks. A philosopher gets hungry and tries to pick up the two chopsticks that are closest to her (the chopsticks that are between her and her left and right neighbors). A philosopher may pick up only one chopstick at a time. When a hungry philosopher has both her chopsticks at the same time, she eats without releasing the chopsticks. When she is finished eating, she puts down both chopsticks and starts thinking again. It is a simple representation of the need to allocate several resources among several processes in a deadlock-free and starvation-free manner.
Solution:One simple solution is to represent each chopstick with a semaphore. A philosopher tries to grab a chopstick by executing a wait() operation on that semaphore. She releases her chopsticks by executing the signal() operation on the appropriate semaphores. Thus, the shared data are

semaphore chopstick[5];

where all the elements of chopstick are initialized to 1. The structure of philosopher is shown

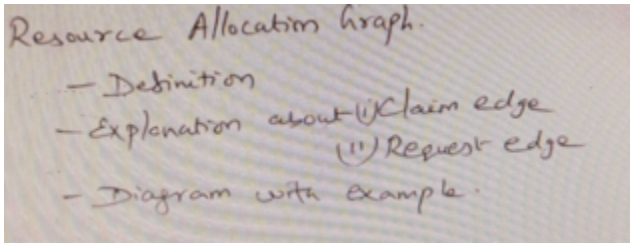

```
while (true)
{
        wait ( chopstick[i] );
        wait ( chopStick[ (i + 1) % 5] );
        // eat
        signal ( chopstick[i] );
        signal (chopstick[ (i + 1) % 5] );
        // think
}
```



Q6b.

Solution:

Resources Allocation Graph



Single Instance of Each Resource Type

• If all resources have only a single instance, then define a deadlock detection algorithm that uses a variant of the resource-allocation graph, called a wait-for graph.

• This graph is obtained from the resource-allocation graph by removing the resource nodes and collapsing the appropriate edges.

• An edge from Pi to Pj in a wait-for graph implies that process Pi is waiting for process Pj to release a resource that Pi needs. An edge Pi → Pj exists in a wait-for graph if and only if the corresponding resource allocation graph contains two edges Pi →Rq and Rq→Pi for some resource Rq.

Example: In below Figure, a resource-allocation graph and the corresponding wait-for graph is presented.

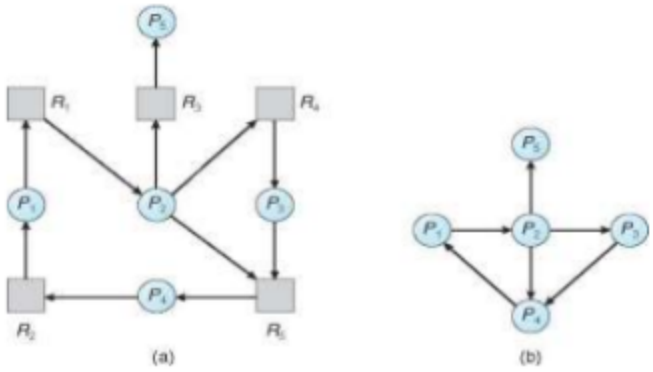


Figure: (a) Resource-allocation graph. (b) Corresponding wait-for graph.

• A deadlock exists in the system if and only if the wait-for graph contains a cycle. To detect deadlocks, the system needs to maintain the wait-for graph and periodically invoke an algorithm that searches for a cycle in the graph.
• An algorithm to detect a cycle in a graph requires an order of n2 operations, where n is the number of vertices in the graph.

Q6c.
Solution:

Need Matrix

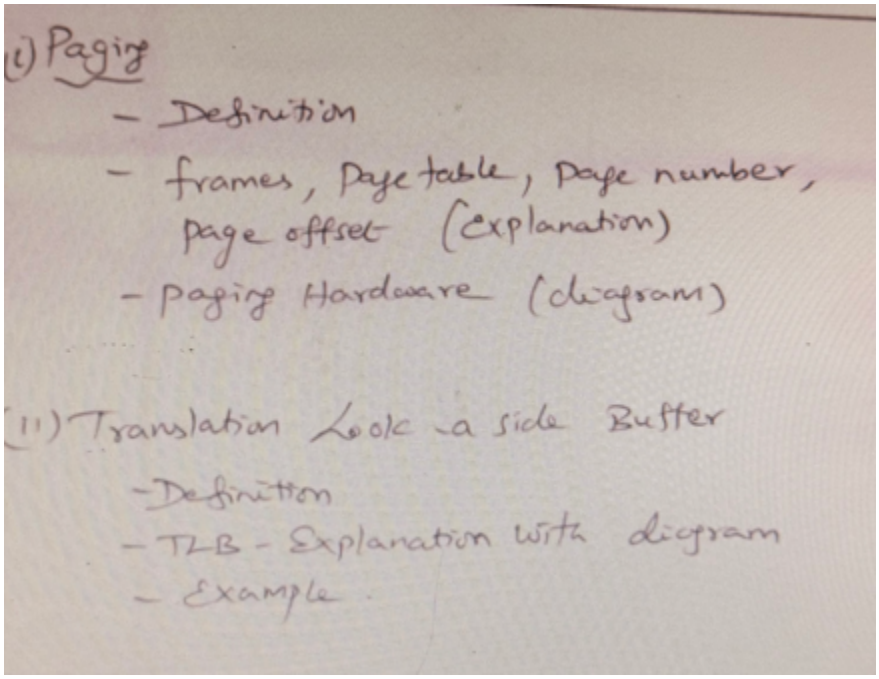
Process	Need			
	A	B	C	D
P_0	0	0	0	0
P_1	0	7	5	0
P_2	1	0	0	2
P_3	0	0	2	0
P_4	0	0	4	2

Safe sequence: P_0, P_2, P_3, P_4, P_1

(10M)

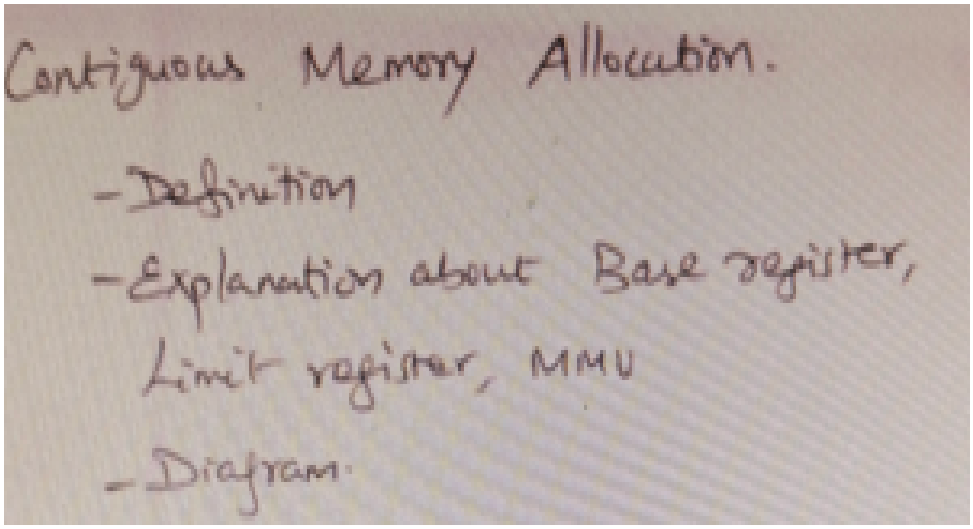
MODULE 3

Q7a.
Solution:



(6M each)

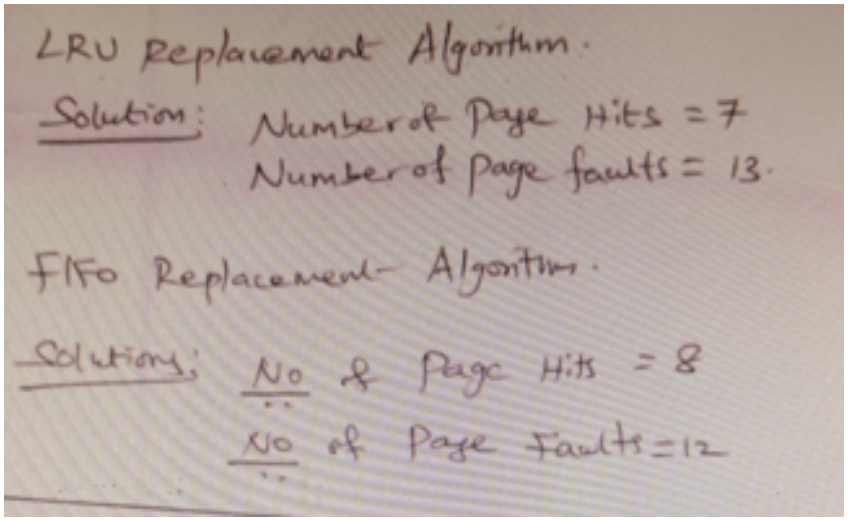
Q7b.
Solution:



(8M)

MODULE 4

Q8a.
Solution:



(10M)

Q8b.
Solution:

• A demand paging is similar to paging system with swapping when we want to execute a process we swap the process the in to memory otherwise it will not be loaded in to memory.
• A swapper manipulates the entire processes, where as a pager manipulates individual pages of the process.
▪ Bring a page into memory only when it is needed
▪ Less I/O needed
▪ Less memory needed
▪ Faster response
▪ More users
▪ Page is needed ⇒ reference to it
▪ invalid reference ⇒abort
▪ not-in-memory ⇒ bring to memory
▪ Lazy swapper– never swaps a page into memory unless page will be needed
▪ Swapper that deals with pages is a pager.
• Basic concept: Instead of swapping the whole process the pager swaps only the necessary pages in to memory. Thus it avoids reading unused pages and decreases the swap time and amount of physical memory needed.
• The valid-invalid bit scheme can be used to distinguish between the pages that are on the disk and that are in memory.
▪ With each page table entry a valid–invalid bit is associated
▪ (v ⇒ in-memory, i⇒not-in-memory)
▪ Initially valid–invalid bit is set to ion all entries
▪ Example of a page table snapshot:

• During address translation, if valid–invalid bit in page table entry is I ⇒ page fault.
• If the bit is valid then the page is both legal and is in memory.
• If the bit is invalid then either page is not valid or is valid but is currently on the disk. Marking a page as invalid will have no effect if the processes never access to that page. Suppose if it access the page which is marked invalid, causes a page fault trap. This may result in failure of OS to bring the desired page in to memory.

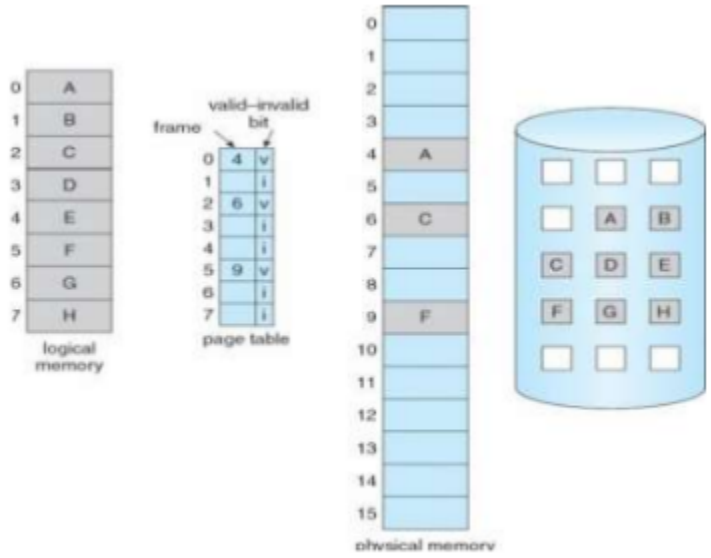
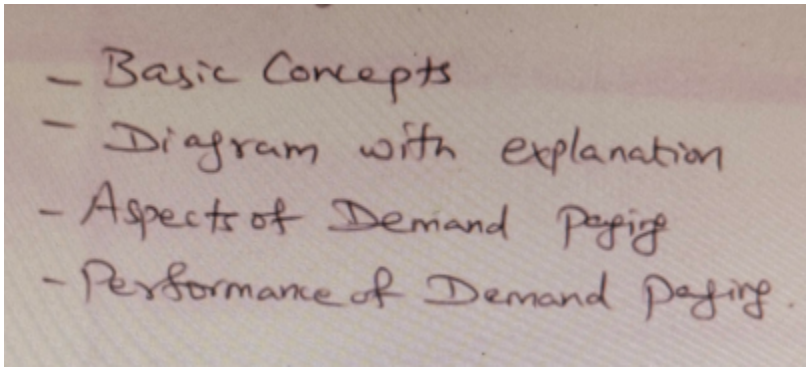


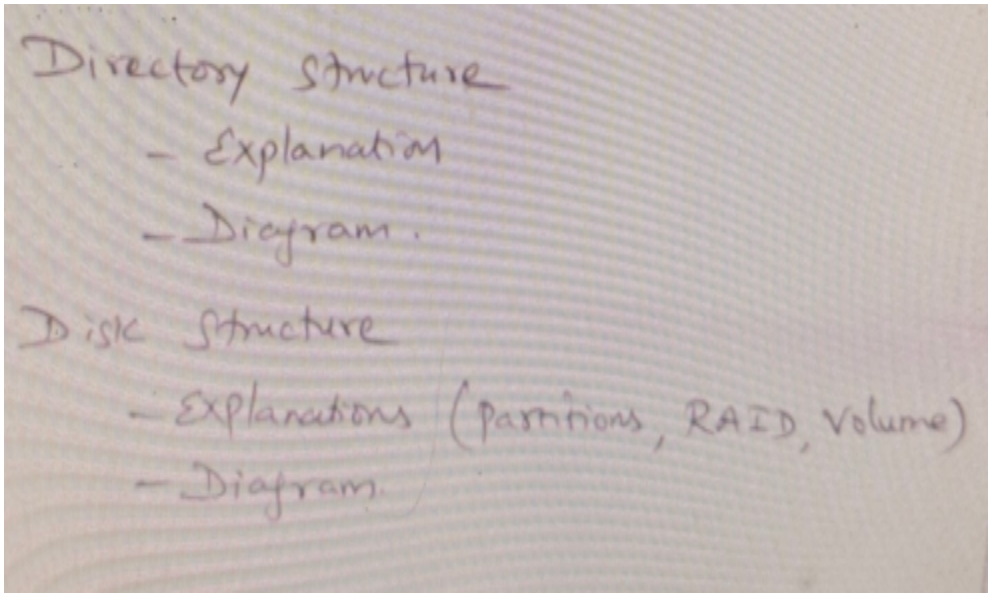
Fig: Page Table when some pages are not in main memory



(2M for each aspect)

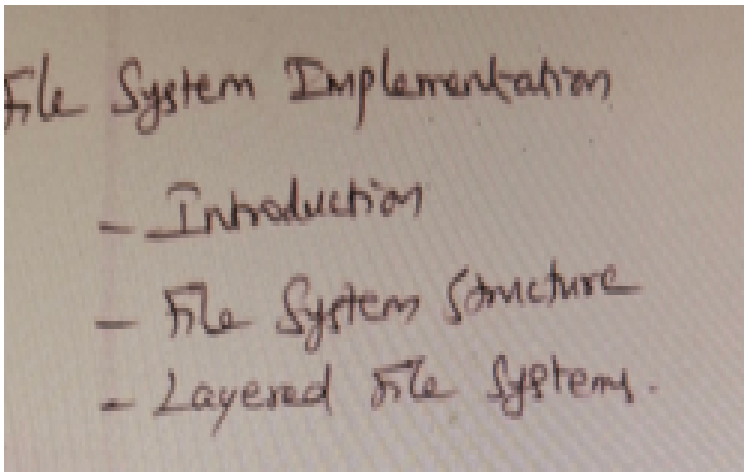
MODULE 5

Q9a.
Solution:



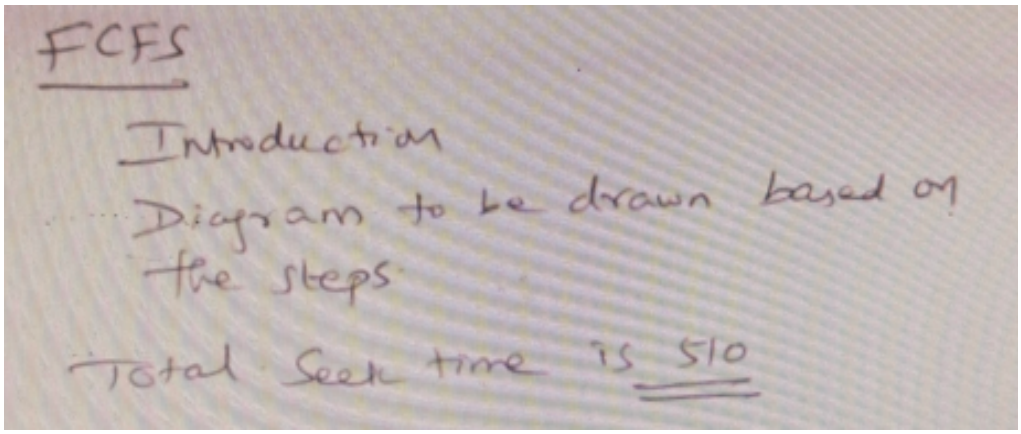
(6M)

Q9b.
Solution:

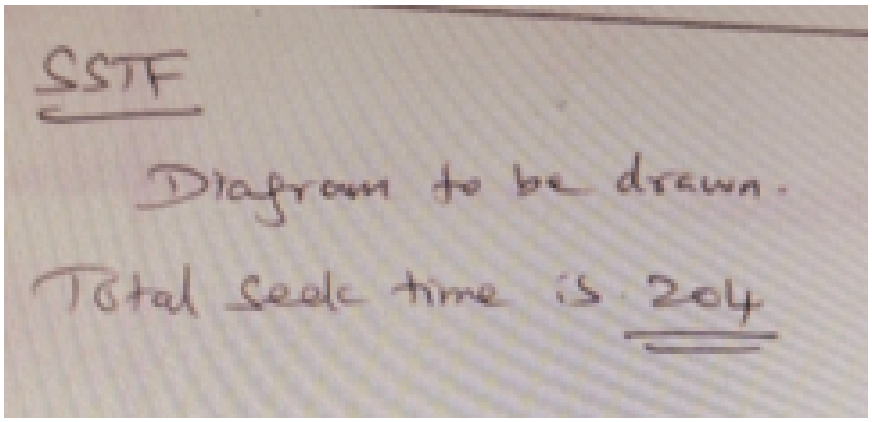


(6M)

Q9c.
Solution:



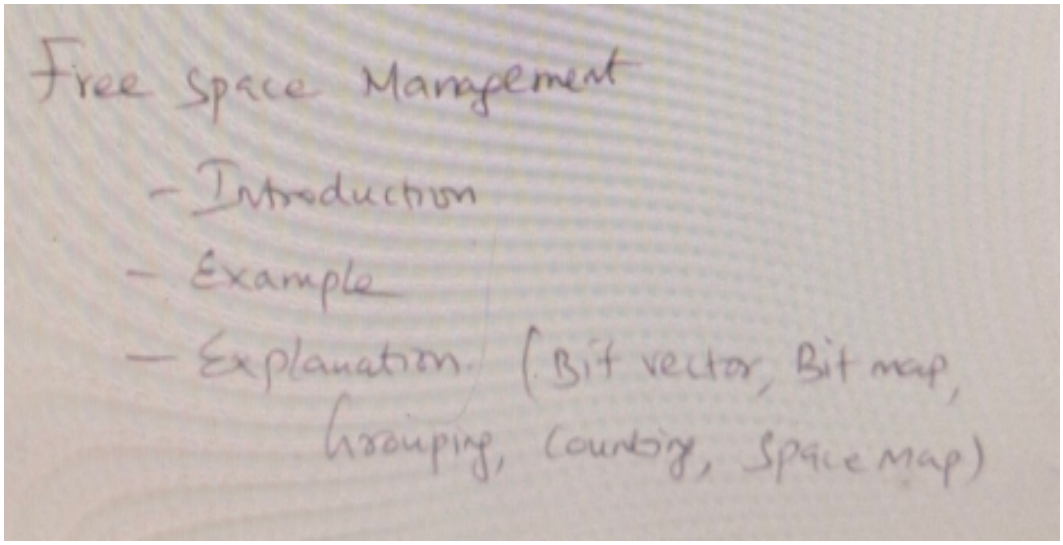
(4M)



(4M)

MODULE 5

Q10a.
Solution:



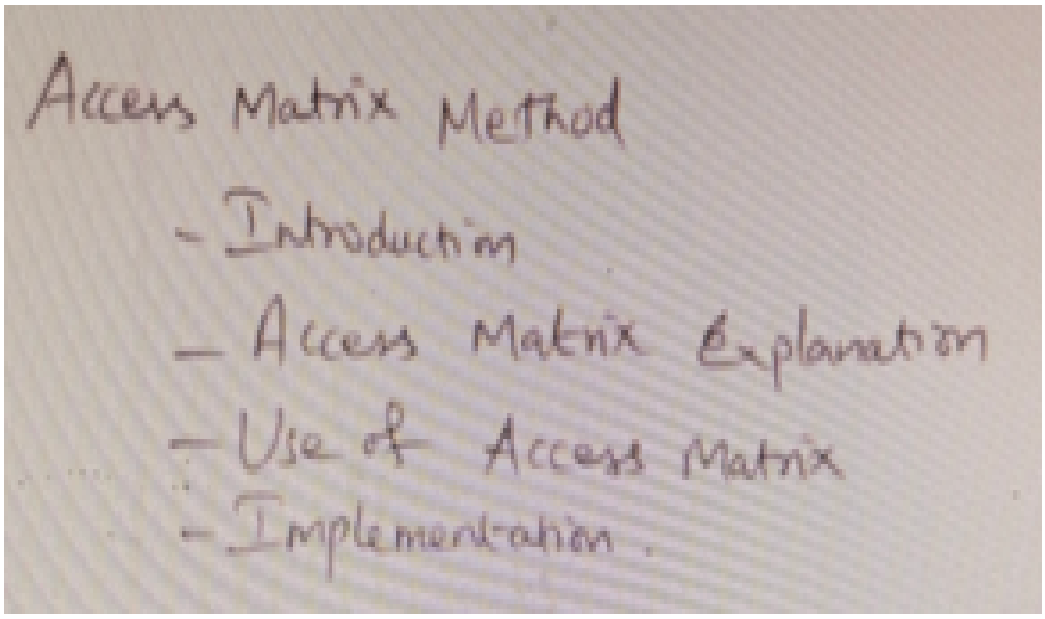
(6M)

Q10b.
Solution:
ACCESS MATRIX
• Our model of protection can be viewed as a matrix, called an access matrix. It is a general model of protection that provides a mechanism for protection without imposing a particular protection policy.
• The rows of the access matrix represent domains, and the columns represent objects.
• Each entry in the matrix consists of a set of access rights.
• The entry access(i,j) defines the set of operations that a process executing in domain Di can invoke on object Oj.

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

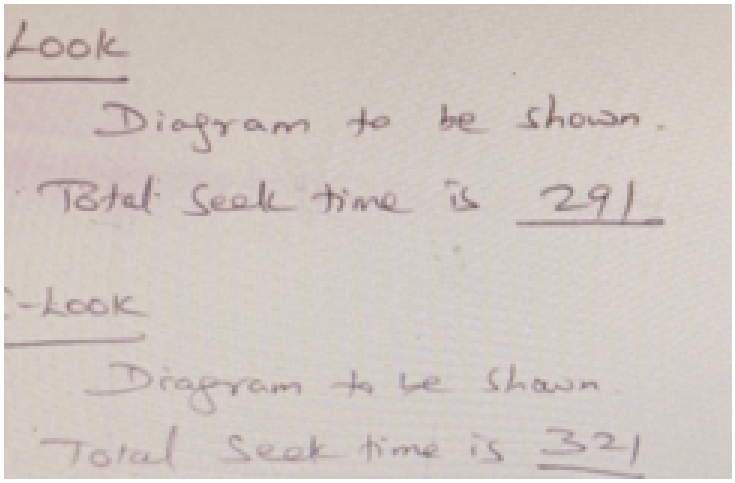
• In the above diagram, there are four domains and four objects—three files (F1, F2, F3) and one printer. A process executing in domain D1 can read files F1 and F3. A process executing in domain D4 has the same privileges as one executing in domain D1; but in addition, it can also write onto files F1 and F3.
• When a user creates a new object Oj, the column Oj is added to the access matrix with the appropriate initialization entries, as dictated by the creator. Domain switching from domain Di to domain Dj is allowed if and only if the access right switch access(i,j). Thus, in the given figure, a process executing in domain D2 can switch to domain D3 or to domain D4. A process in domain D4 can switch to D1, and one in domain D1 can switch to domain D2.

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			



(6M)

Q10c.
Solution:



(4M each)