FORMAT-1

Semester: 4th Semester.
Programme:B.Tech.
Branch/Specialization: SCE.

SPRING END SEMESTER EXAMINATION-2023

Spring Semester, 2023 (Programme)

SUBJECT: OPERATING SYSTEMS

CODE: CS2002

(For 2021 Admitted Batches)

Time: 3 Hours **Full Marks: 50**

Answer any SIX questions.
Question paper consists of four SECTIONS i.e. A, B, C and D.
Section A is compulsory.
Attempt minimum one question each from Sections B, C, D.
The figures in the margin indicate full marks.
Candidates are required to give their answers in their own words as far as practicable and all parts of a question should be answered at one place only.

SECTION-A

1 Answer the following questions.

a The Shortest Job First (SJF) process scheduling algorithm leads to the problem of starvation. Discuss some of the ways to resolve this issue.

Ans:

Starvation in SJF exist only if processes with Lower Burst Time appears in queue before the process with Higher Burst time is executed. Since the algorithm will always choose the process with lowest Burst Time, the process with higher Burst Time will never be able to get the share of CPU. It is difficult to find a solution to starvation problem in SJF scheduling algorithm.

One possible solution to this problem is to avoid more process to be scheduled in job pool once it is full i.e., Scheduler must wait until job pool is completely empty before it can get more process for execution.

b There are some similarities between Semaphores and condition variables (of Monitor). What are the major differences between semaphores and condition variables?

Ans:

Does not allow threads to wait. Each thread keeps running and last thread sets the semaphore value to zero and goes to sleep.	Allows threads to wait until particular condition occurs.
Used to solve problem of some critical sections in process synchronization.	Used with mutex to signal changing states from one thread to another.
Main aim is to control access to common resource by multiple processes and avoid critical section problem in concurrent system like multitasking operating system.	Main aim is to support operations that wake one or wake all waiting threads.
Can be used anywhere except in a monitor.	Can only be used in monitors.
wait() does not always block its caller.	wait () usually blocks its caller always.
They are sticky as they have memory and sem_post() will increment semaphore even if no one has called sem_wait().	They are non-sticky as signal () is not saved if there is no one waiting for signal ().
It is simply a counter + mutex + wait queue.	It is simply a wait-queue.
It can be used as a conditional variable and therefore is more lightweight and flexible.	It cannot be used as a semaphore and therefore is not flexible.
Signals are not lost even if there is nobody waiting on the queue.	Signals are lost if there is nobody waiting on the queue.
In this, we cannot unlock all waiting threads in same instant using broadcast unlocking.	In this, we can unlock all waiting threads in same instant using broadcast unlocking.

c Explain the difference between the logical/virtual and physical address. Discuss the benefits of virtual memory.

Ans:

Logical Address	Physical Address
Generated by the CPU.	A location in the memory unit.
The address space consists of the set of all logical addresses.	This address is a set of all physical addresses.
Generated by CPU with reference to a specific program.	Computed using Memory Management Unit (MMU).
User has the ability to view the logical address of a program.	User can't view the physical address of program directly.
User can use the logical address in order to access the physical address.	User can indirectly access the physical address.

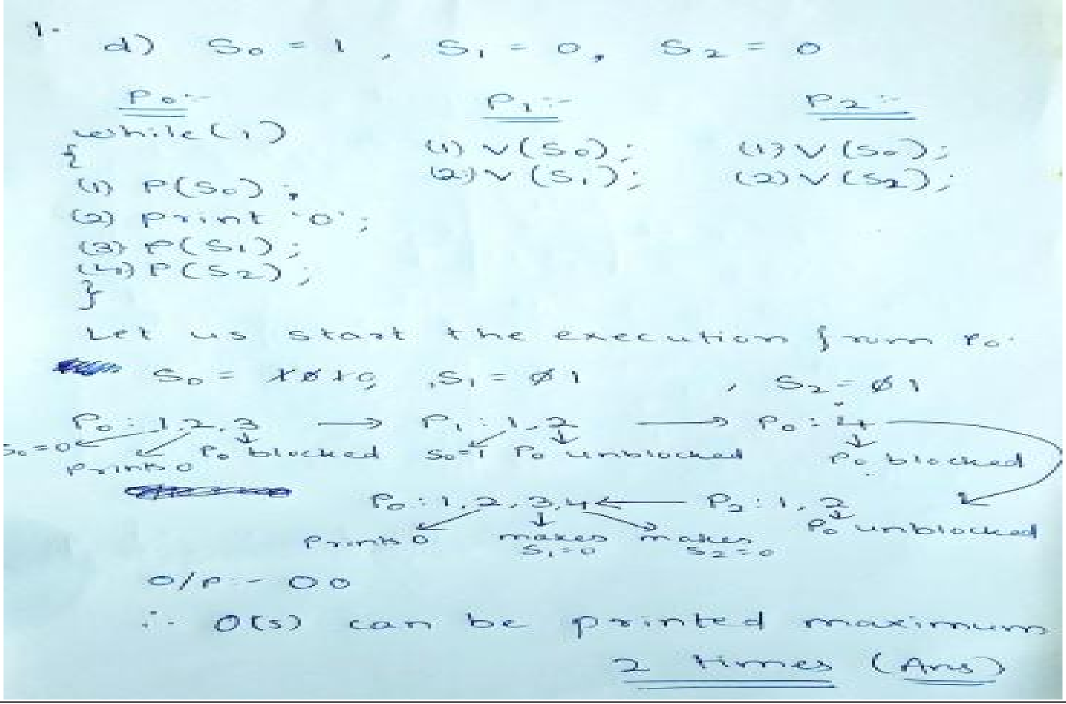
d The code for three concurrent processes P0, P1 and P2is as follows:

Process P0	Process P1	Process P2

while(1) {
P(S0);
print '0';
P(S1);
P(S2); }
	V(S0);
	V(S1);
		V(S0);
		V(S2);

The binary semaphoresS0, S1 and S2 in the above code are initialized to 1, 0 and 0 respectively. What is the maximum number of zeroes to be printed? justify.

Ans:



1- d) $S_0 = 1$, $S_1 = 0$, $S_2 = 0$

```
P0:-                P1:-                P2:-
while(1)            (1) V(S0);          (1) V(S0);
{                   (2) V(S1);          (2) V(S2);
(1) P(S0);
(2) print '0';
(3) P(S1);
(4) P(S2);
}
```

Let us start the execution from P0.

$S_0 = \cancel{1}\,\cancel{0}\,1\,0$, $S_1 = \cancel{0}\,1$, $S_2 = \cancel{0}\,1$

P0: 1,2,3 ⟶ P1: 1,2 ⟶ P0: 4

S0=0, print 0, P0 blocked; S0=1, P0 unblocked; P0 blocked

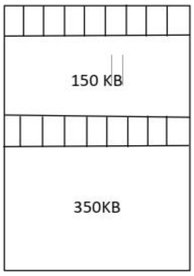
P0: 1,2,3,4 ⟵ P2: 1,2

print 0, makes S1=0, makes S2=0; P0 unblocked

o/p - 00

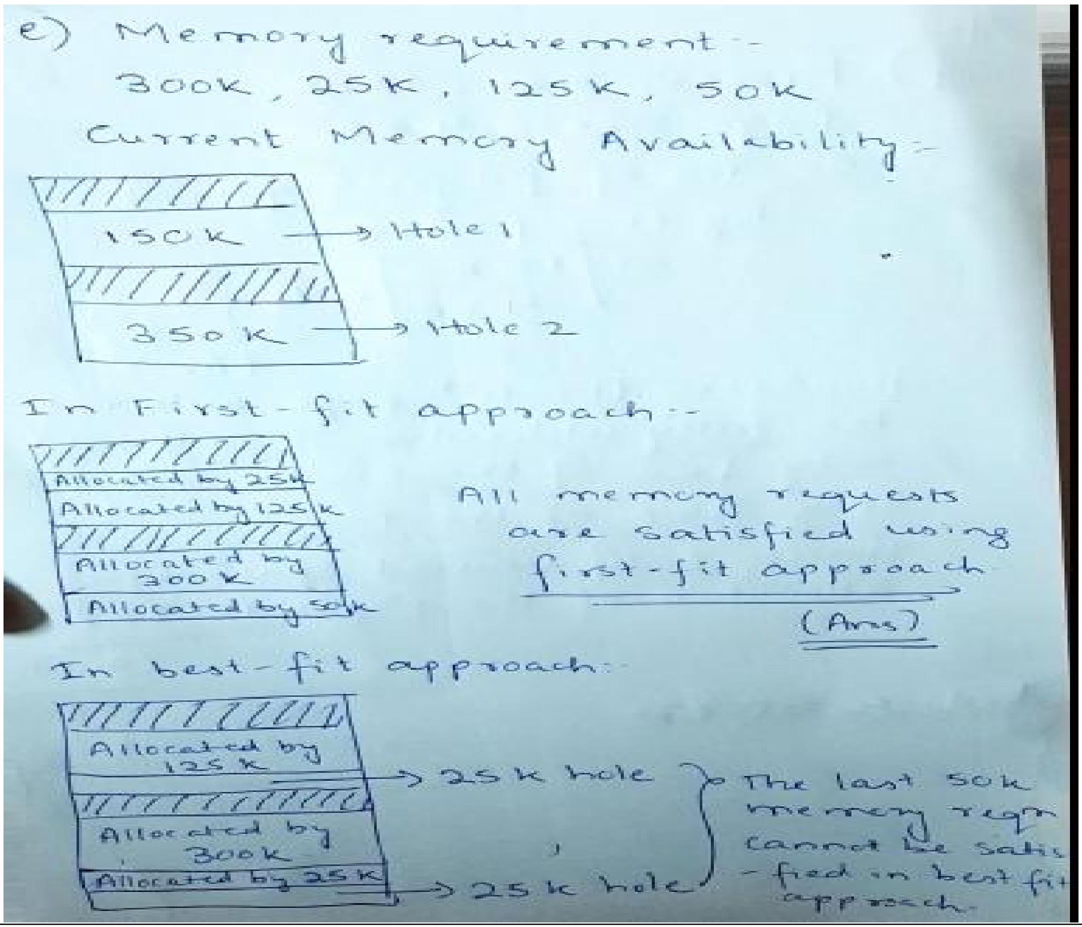
∴ 0(s) can be printed maximum 2 times (Ans)

e If the memory requests from processes are in the order: 300K, 25K, 125K and 50K and the current memory availability is as shown below:



Then which approach, between "first fit" and "best fit", can satisfy the above memory requests?

Ans:



e) Memory requirement :-
300K, 25K, 125K, 50K

Current Memory Availability :-

150K → Hole 1
350K → Hole 2

In First-fit approach :-

Allocated by 25K
Allocated by 125K
Allocated by 300K
Allocated by 50K

All memory requests are satisfied using first-fit approach (Ans)

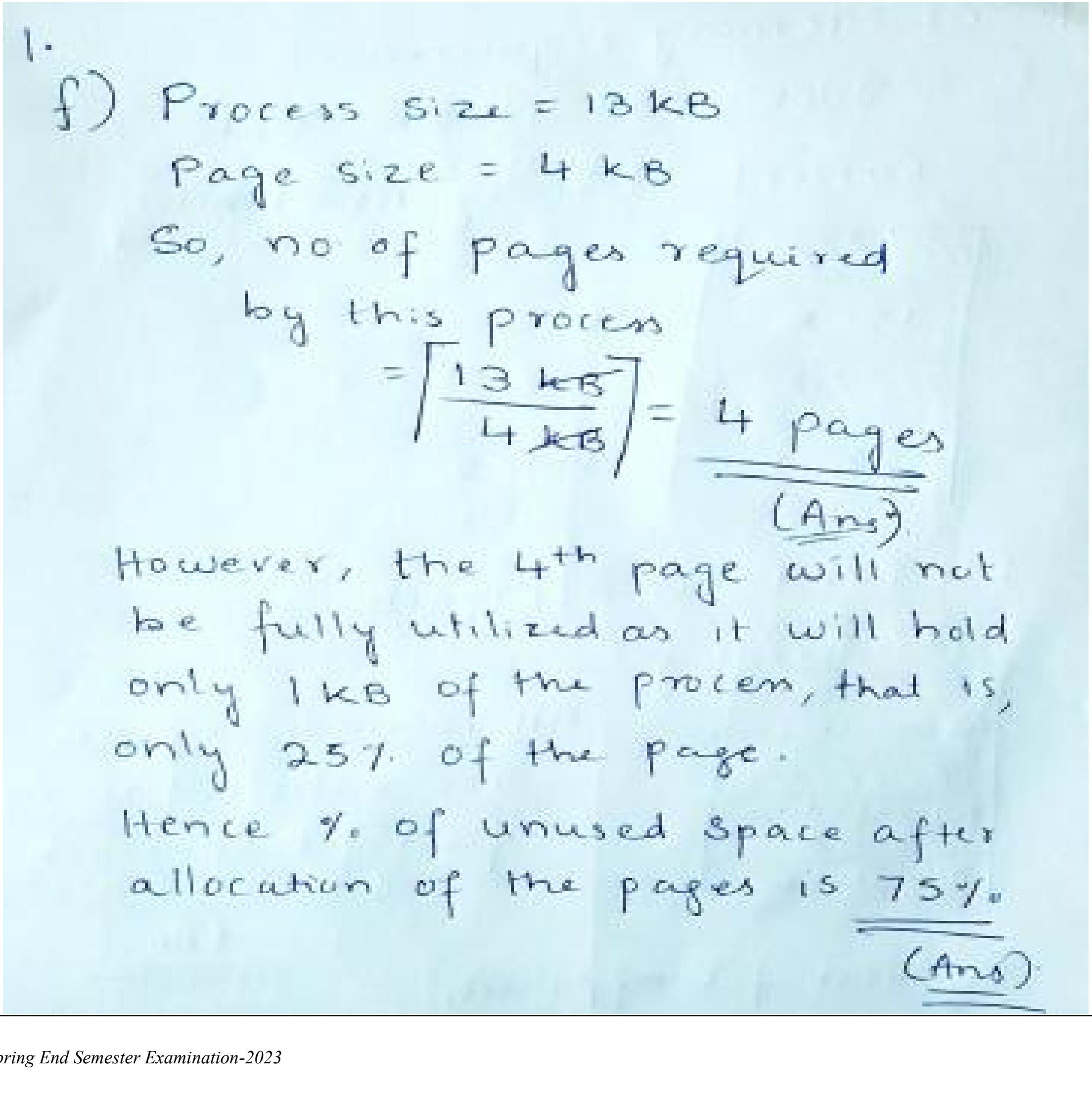
In best-fit approach :-

Allocated by 125K
→ 25K hole
Allocated by 300K
Allocated by 25K
→ 25K hole

The last 50K memory reqn cannot be satis-fied in best fit approach.

f Consider a process of size 13KB and page size is 4KB. How many pages would this process need? Calculate the % of unused space after allocation of the required pages.

Ans:



1.

f) Process size = 13 KB

Page size = 4 KB

So, no of pages required by this process

$= \left\lceil \frac{13 \cancel{KB}}{4 \cancel{KB}} \right\rceil = \underline{\underline{4 \text{ pages}}}$ (Ans)

However, the 4th page will not be fully utilized as it will hold only 1 KB of the process, that is, only 25% of the page.

Hence % of unused space after allocation of the pages is 75%. (Ans) g A Resource Allocation Graph (RAG) contains a directed cycle. Then, under what circumstances is there a possibility of deadlock?

Ans:

If there is only one unit per resource type, there is a deadlock.

Otherwise, a deadlock is only a possibility.

h When a detection algorithm determines a deadlock has occurred? What are some ways to recover the system from this deadlock?

Ans:

Process Termination, Resource Preemption. Explain each of them.

i Rotational latency = 0.5 × time to take one rotation true or false. Justify your answer.

Ans:

True.

The average rotational latency for a disk is half the amount of time it takes for the disk to make one revolution.

j Let a Hard Disk has 6 surfaces, each surface has 8 tracks, and each track has 16 sectors per track. Find the cylinder, head and sector address of the logical block address 610.

Ans:

<6, 2, 2>

1 cylinder = 6 × 16 = 96 sectors

610/96=6th cylinder

Remainder = 32, advance 2 surfaces, 2 sector.

SECTION-B (Learning levels 1, 2, and 3)

2 a Write the pseudo code for Peterson solution to the critical section problem. Explain the correctness of the solution which satisfies the 3 conditions for critical section problem. 4

Ans:

```
P(i) {
        flag[i] = TRUE;
        turn = j;
        while(flag[j] and turn=j);
        critical Section
        flag[i] := FALSE;
        Remainder Section
}
```

Mutual exclusion is preserved: (Proof by contradiction)

Assume both processes Pi and Pj are in their CS.

Each Pi enters its critical section only if either flag[j] = false or turn = i.

P0 and P1 are both in CS only if flag[0] = flag[1] = true and turn = i for each Pi (which is impossible), turn cannot be both values i and j at a time, therefore one process must have entered its CS first (without loss of generality, say P0), but this means that P1 could not have found turn = 1 and therefore could not have entered its CS (i.e. contradiction).

Progress requirement is satisfied:

Pi cannot enter CS only if stuck in while() with condition flag[j] = true and turn = j.

Case-I: If Pj is not ready to enter CS then flag[j] = false and Pi can then enter its CS

Case-II: If Pj is ready to enter CS then Pj has set flag[j]=true and is in its while(), then either turn=i or turn=j If turn=i, then Pi enters CS.

If turn=j then Pj enters CS, will reset flag[j]=false on exit: allowing Pi to enter CS

Bounded waiting requirements is satisfied:

P1 is executing its CS repeatedly upon exiting its CS, P1 sets flag[1] = false,

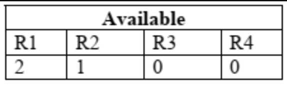
hence the while loop is false for P0 and it can go.

However, P1 may attempt to re-enter its CS before P0 has a chance to run

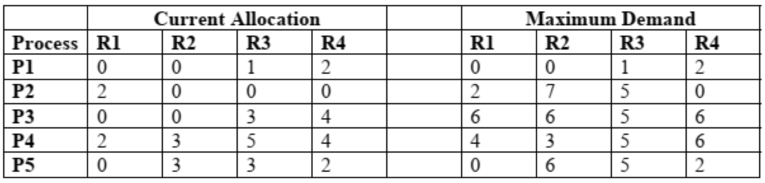
but to re-enter, P1 sets flag[1]=true and sets turn=0, hence the while loop is true for P1 and it waits but the while loop is now false for P0 and it can enter into CS at most one CS entry by P0 (bounded waiting).

.

b



Available			
R1	R2	R3	R4
2	1	0	0



	Current Allocation					Maximum Demand			
Process	R1	R2	R3	R4		R1	R2	R3	R4
P1	0	0	1	2		0	0	1	2
P2	2	0	0	0		2	7	5	0
P3	0	0	3	4		6	6	5	6
P4	2	3	5	4		4	3	5	6
P5	0	3	3	2		0	6	5	2

4

Consider the above snapshot of a system. There are no current outstanding queued unsatisfied requests.
Is this system currently in safe or unsafe? Justify?
If safe write down the safe sequence. If unsafe mention the processes in unsafe state.
If a request from P3 arrives for (0, 1, 0, 0), can that request be safely granted immediately? In what state (deadlock, safe, unsafe) would immediately granting that whole request leave the system? Which processes, if any, are or may become deadlocked if this whole request is granted immediately?
Ans:
(i) & (ii)

Need Matrix				
	R1	R2	R3	R4
P1	0	0	0	0
P2	0	7	5	0
P3	6	6	2	2
P4	2	0	0	2
P5	0	3	2	0

(iii)

Allocation Matrix					Need Matrix				Available			
	R1	R2	R3	R4	R1	R2	R3	R4	R1	R2	R3	R4
P1	0	0	1	2	0	0	0	0	2	0	0	0
P2	2	0	0	0	0	7	5	0				
P3	0	1	3	4	6	5	2	2				
P4	2	3	5	4	2	0	0	2				
P5	0	3	3	2	0	3	2	0				

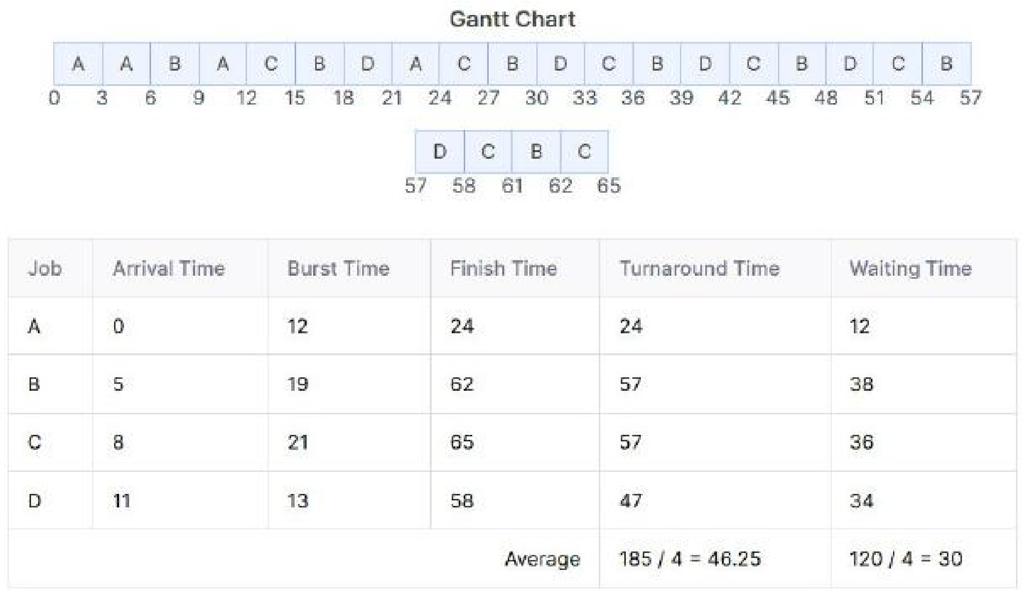
Allocation, available and need matrix: 1 mark
Procedure (explanation step by step): 1 mark
Unsafe state. Now P1, P4, P5 can finish, but with available now (4, 6, 9, 8) neitheP2 nor P3 can be satisfied.

3 a

4

Process	Arrival Time	Execution Time
P1	0	12
P2	5	19
P3	8	21
P4	11	13

The above Table gives the snapshot of the processes in the system with arrival time and execution time in milliseconds. Draw a Gantt chart showing the scheduling of the processes using Round Robin scheduling algorithm. Find the waiting time of each of the processes. Also calculate the average waiting time. (Given time quantum = 3 milliseconds)
Ans:



Gantt Chart

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

0 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57

D	C	B	C

57 58 61 62 65

Job	Arrival Time	Burst Time	Finish Time	Turnaround Time	Waiting Time
A	0	12	24	24	12
B	5	19	62	57	38
C	8	21	65	57	36
D	11	13	58	47	34
			Average	185 / 4 = 46.25	120 / 4 = 30

b Explain the various stages of Process State Transition with a neat diagram. 4

Ans:

Each process may be in any one of the following states:

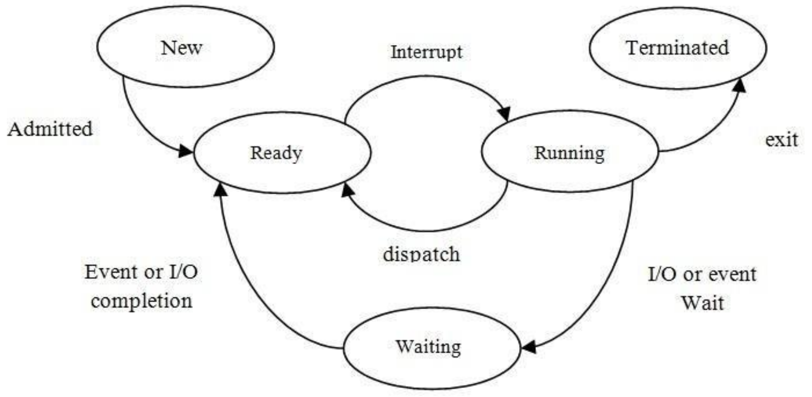
New: Program going to be picked up by the OS into the main memory is called a new process i.e. a Process being created.

Ready: New process after being picked up by OS from the secondary memory is put in the main memory for execution i.e. the state of the process which is ready for the execution and resides in the main memory.

Running: One of the processes from the ready state will be chosen by the OS depending upon the scheduling algorithm to be allocated to CPU for execution.

Waiting: process waits for a certain resource to be assigned or for the input from the user then the OS move this process to the block or wait state and assigns the CPU to the other processes.

Terminated: process finishes its execution, then it comes in the termination state. All the context of the process (Process Control Block) will also be deleted and the process will be terminated by the Operating system.



SECTION-C (Learning Levels 3 and 4)

4 a Consider the page reference string: 2, 3, 1, 2, 4, 5, 1, 2, 3, 4, 5, 2, 6, 2, 1, 2, 3, 1, 4, 5, 6. 4

How many page faults would occur for the following page fault algorithms, assuming 3 and 4 available frames. Initially all the frames are empty.

Optimal replacement

LRU replacement

Ans:

(i) Optimal 12 and 9 (ii) LRU 17 and 14

b Implement Reader-Writer problem with reader dominance using Critical region and Monitor. (Write the Pseudocode) 4

Ans:

Code for Writer Process	Code for Reader Process

wait(write);
Write into the file
signal(wrt);
	int readcount=0;
	wait(mutex);
	readcount++;
	if(readcount==1) wait(wrt);
	signal(mutex);
	Read from file
	wait(mutex);
	readcount--;
	if (readcount == 0) signal(wrt);
	signal(mutex);

Monitor:

```
monitor ReadersWriters
        condition OKtoWrite, OKtoRead;
        int ReaderCount = 0;
        Boolean busy = false;

        procedure StartRead() {
                if(busy) OKtoRead.wait;
                ReaderCount++;
                OKtoRead.signal();
        }

        procedure EndRead() {
                ReaderCount-- ;
                if(ReaderCount==0) OKtoWrite.signal();
        }

        procedure StartWrite() {
                if(busy || ReaderCount!=0) OKtoWrite.wait();
                busy = true;
        }

        procedure EndWrite() {
                busy = false;
                If(OKtoRead.Queue) OKtoRead.signal();
                else OKtoWrite.signal();
        }

        Reader() {
                while(TRUE) {
                        ReadersWriters.StartRead();
                        readDatabase();
                        ReadersWriters.EndRead();
                }
        }

        Writer() {
                while (TRUE) {
                        make_data(&info);
                        ReaderWriters.StartWrite();
                        writeDatabase();
                        ReadersWriters.EndWrite();
                }
        }
```

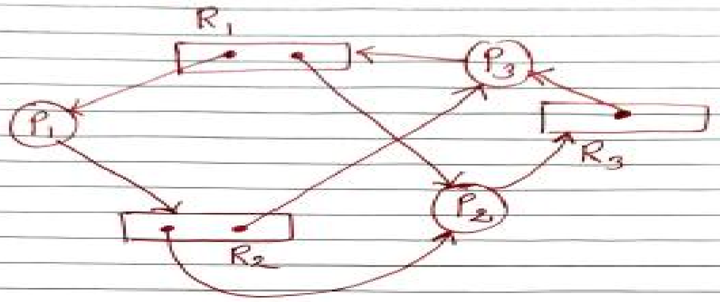
5	a	A system has three processes (P1, P2, P3) and three resource types (R1, R2, R3). There are two instances of R1 and R2 and one instance of R3. P1 holds a R1 and is requesting a R2. P2	4

holds a R1 and a R2 is requesting an R3. P3 holds a R2 and R3 and is requesting a R1. Draw the resource allocation graph for this situation. Does the deadlock situation exist? Prove your answer using Banker's Algorithm.

Ans:

Note:- As the System is Unsafe, If we proceed with this Deadlock may occur.

Q. 5(a)



	Allocation			Need/Request			TOTAL		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	1	0	0	0	1	0	2	2	1
P2	1	1	0	0	0	1			
P3	0	1	1	1	0	0			
	2	2	1						

Available → [TOTAL − Allocation]

R1	R2	R3
0	0	0

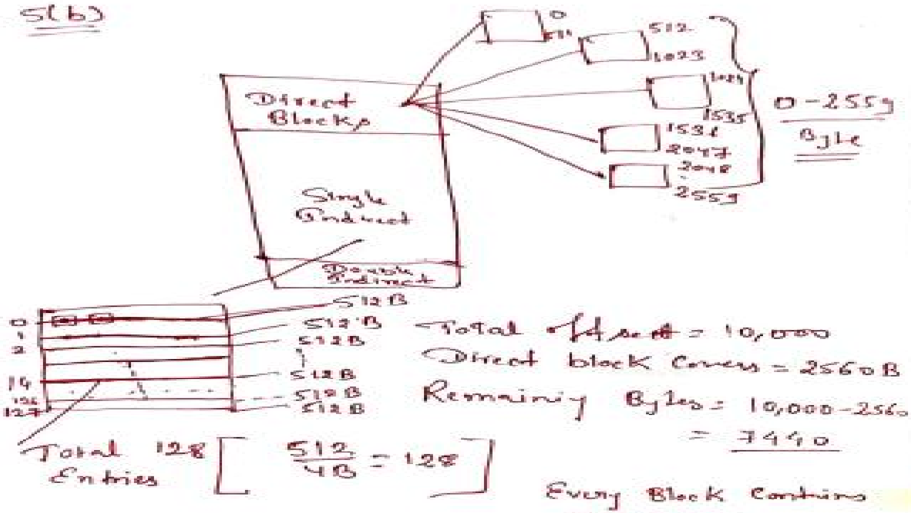
Unsafe

With the available resources, We can not satisfy the need of any Process. So, no Safe Sequence.

b Consider a file system with 5 direct disk blocks, 1 single indirect block and 1 double indirect block. Size of disk block in 512 bytes. 4 bytes are required to store one block number. If a process makes a request to access a byte at offset of 10000, find out which index block contain this byte information. 4

Ans:

Q. 5(b)



$\left[\frac{512}{4B} = 128 \right]$

Total offset = 10,000

Direct block Covers = 2560B

Remaining Bytes = 10,000 − 2560

= 7440

Every Block Contains 512 Bytes

∴ for 7440 Bytes Block ??

$\frac{7440}{512} = 14.53$

⇓

15th Block

Total 128 block

Index ⇒ 0 1 2 3 . . . $\boxed{14}$ 15 16 127

↳ $\boxed{\text{15th Block}}$ of Single indirect.

Contains ~~address~~ Byte offset 10,000.

6 a What do you mean by multilevel feedback scheduling? Briefly describe its characteristics, features, pros and cons of the scheduling algorithm. Explain the algorithm by using a neat diagram with a suitable example. 4

Ans:

Multilevel feedback scheduling is a CPU scheduling algorithm that allows a process to be moved between different priority queues. Processes are initially placed in a high-priority queue and are moved to a lower-priority queue after a certain amount of time. If a process uses more CPU time than expected, it can be moved to a lower priority queue, while a process that waits for an I/O operation to complete can be moved to a higher priority queue.

Characteristics:

Multilevel feedback scheduling is a preemptive approach.
It allows the priority of a process to change during its execution.
It supports multiple queues with different priorities.
It can handle both CPU-bound and I/O-bound processes.

Features:

Multilevel feedback scheduling is a dynamic algorithm that adapts to changes in the workload.
It allows for better response time for interactive processes, as they can be given higher priority and moved to the higher priority queue.
The implementation of multiple queues with different priorities allows for better usage of system resources.
The algorithm is able to handle a mix of both short and long running processes.

Pros:

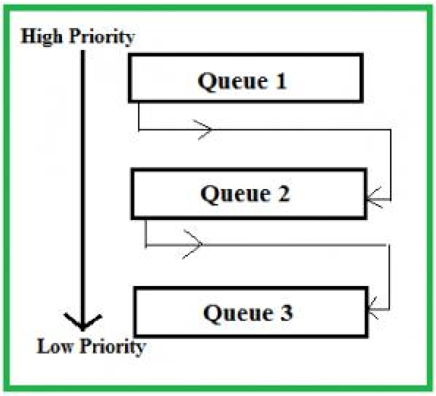
Multilevel feedback scheduling provides a good balance between performance and fairness in CPU scheduling.
It is able to handle both CPU and I/O intensive tasks well.
The use of multiple queues allows for better performance and resource utilization.

Cons:

Multilevel feedback scheduling can be complex to implement.
The parameter settings for the algorithm can greatly affect its performance.
The algorithm may be less efficient than other algorithms for certain types of workloads.

Example:

Suppose we have three queues: Queue1, Queue2, and Queue3. Queue1 has the highest priority and uses round-robin scheduling with a time quantum of 8. Queue2 uses priority scheduling, and Queue3 uses FCFS (First Come First Serve) scheduling. A new process enters the system and is placed in Queue1. It receives 8 milliseconds of CPU time, but it does not complete in that time. Therefore, it is moved to Queue2. After some time, it is demoted to Queue3, where it completes its execution. Meanwhile, another process enters the system and is placed in Queue1. It completes its execution in one round-robin time quantum, so it exits the system. Finally, a third process enters the system and is placed in Queue1. It receives 8 milliseconds of CPU time, but it does not complete in that time. Therefore, it is moved to Queue2. It continues to run in Queue2 until it completes its execution.



b There is a memory with four partitions 4K, 8K, 20K, and 2K (in order). The eight processes come at time 0 ms with different request size and the execution time (ms) as given in the following table. 4

Process no.	P1	P2	P3	P4	P5	P6	P7	P8
Req. Size	2K	14K	3K	6K	6K	10K	7K	4K
Execution time	4	9	2	8	1	6	5	3

Calculate the time at which process P7 will be completed if Best-Fit method is used in fixed partition memory allocation. Explain with proper diagram.

Ans:

time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4k	P3	P3	P8	P8	P8										
8k	P4	P4	P4	P4	P4	P4	P4	P4	P5	P7	P7	P7	P7	P7	
20k	P2	P2	P2	P2	P2	P2	P2	P2	P2	P6	P6	P6	P6	P6	P6
2k	P1	P1	P1	P1											

So P7 will be completed at 14ms.

SECTION-D (Learning levels 4, 5, 6)

7 a Disk requests come into the disk driver for cylinders: 10, 22, 55, 2, 28, 12, 6, 40 and 38 in that order. The disk drive has 60 cylinders numbered 0 to 59 and the current head position is 26. A seek takes 6 millisecond / cylinder moved. How much seek time is needed for: 4

(A) FCFS (B) SSTF

(C) SCAN (D) LOOK

Ans:

(A) In the FCFS (First-Come-First-Serve) scheduling algorithm, disk requests are serviced in the order they arrive. Given the current head position is 26 and the requests for cylinders are 10, 22, 55, 2, 28, 12, 6, 40, and 38, we can calculate the seek time as follows:

Move from cylinder 26 to 10: abs(26 - 10) = 16 cylinders
Move from cylinder 10 to 22: abs(10 - 22) = 12 cylinders
Move from cylinder 22 to 55: abs(22 - 55) = 33 cylinders
Move from cylinder 55 to 2: abs(55 - 2) = 53 cylinders
Move from cylinder 2 to 28: abs(2 - 28) = 26 cylinders
Move from cylinder 28 to 12: abs(28 - 12) = 16 cylinders
Move from cylinder 12 to 6: abs(12 - 6) = 6 cylinders
Move from cylinder 6 to 40: abs(6 - 40) = 34 cylinders
Move from cylinder 40 to 38: abs(40 - 38) = 2 cylinders
Total cylinders moved: 16 + 12 + 33 + 53 + 26 + 16 + 6 + 34 + 2 = 198 cylinders
Given that a seek takes 6 ms per cylinder moved, the total seek time for FCFS would be:
198 cylinders * 6 ms/cylinder = 1188 ms

(B) The SSTF (Shortest Seek Time First) scheduling algorithm services disk requests in the order of the shortest seek time from the current head position. Given the current head position is 26 and the requests for cylinders are 10, 22, 55, 2, 28, 12, 6, 40, and 38, we can calculate the seek time as follows:

Move from cylinder 26 to 28: abs(26 - 28) = 2 cylinders
Move from cylinder 28 to 22: abs(28 - 22) = 6 cylinders
Move from cylinder 22 to 12: abs(22 - 12) = 10 cylinders

Move from cylinder 12 to 10: abs(12 - 10) = 2 cylinders
Move from cylinder 10 to 6: abs(10 - 6) = 4 cylinders
Move from cylinder 6 to 2: abs(6 - 2) = 4 cylinders
Move from cylinder 2 to 38: abs(2 - 38) = 36 cylinders
Move from cylinder 38 to 40: abs(38 - 40) = 2 cylinders
Move from cylinder 40 to 55: abs(40 - 55) = 15 cylinders
Total cylinders moved: 2 + 6 + 10 + 2 + 4 + 4 + 36 + 2 + 15 = 81 cylinders
Given that a seek takes 6 ms per cylinder moved, the total seek time for SSTF would be:
81 cylinders * 6 ms/cylinder = 486 ms

(C) The SCAN scheduling algorithm services disk requests by moving the disk arm towards one end of the disk until it reaches the end, then reversing the direction and servicing the remaining requests as it moves in the opposite direction. Given the current head position is 26 and the requests for cylinders are 10, 22, 55, 2, 28, 12, 6, 40, and 38, we can calculate the seek time as follows:
2, 6, 10, 12, 22, 28, 38, 40, 55
Now we'll separate the requests into two groups based on the current head position (26):
Left: 2, 6, 10, 12, 22
Right: 28, 38, 40, 55
The SCAN algorithm will first service the right group because, head movement direction is not there and for this we have to choose the shortest distance and move in that direction. Then go all the way to the end (cylinder 59), and then reverse direction to service the left group:
Move from cylinder 26 to 28: abs(26 - 28) = 2 cylinders
Move from cylinder 28 to 38: abs(28 - 38) = 10 cylinders
Move from cylinder 38 to 40: abs(38 - 40) = 2 cylinders
Move from cylinder 40 to 55: abs(40 - 55) = 15 cylinders
Move from cylinder 55 to 59: abs(55 - 59) = 4 cylinders (reaching the end of the disk)
Move from cylinder 59 to 22: abs(59 - 22) = 37 cylinders
Move from cylinder 22 to 12: abs(22 - 12) = 10 cylinders
Move from cylinder 12 to 10: abs(12 - 10) = 2 cylinders
Move from cylinder 10 to 6: abs(10 - 6) = 4 cylinders
Move from cylinder 6 to 2: abs(6 - 2) = 4 cylinders
Total cylinders moved: 2 + 10 + 2 + 15 + 4 + 37 + 10 + 2 + 4 + 4 = 90 cylinders
Given that a seek takes 6 ms per cylinder moved, the total seek time for SCAN would be:
90 cylinders * 6 ms/cylinder = 540 ms

(D) The LOOK scheduling algorithm is similar to SCAN, but instead of going all the way to the end of the disk, it only goes as far as the last requested cylinder in each direction before reversing. Given the current head position is 26 and the requests for cylinders are 10, 22, 55, 2, 28, 12, 6, 40, and 38, we can calculate the seek time as follows:
We'll first sort the requests in ascending order:
2, 6, 10, 12, 22, 28, 38, 40, 55
Now we'll separate the requests into two groups based on the current head position (26):
Left: 2, 6, 10, 12, 22
Right: 28, 38, 40, 55
The LOOK algorithm will first service the right group because, head movement direction is not there and for this we have to choose the shortest distance and move in that direction. Then reverse direction to service the left group:
Move from cylinder 26 to 28: abs(26 - 28) = 2 cylinders
Move from cylinder 28 to 38: abs(28 - 38) = 10 cylinders
Move from cylinder 38 to 40: abs(38 - 40) = 2 cylinders
Move from cylinder 40 to 55: abs(40 - 55) = 15 cylinders
Move from cylinder 55 to 22: abs(55 - 22) = 33 cylinders
Move from cylinder 22 to 12: abs(22 - 12) = 10 cylinders
Move from cylinder 12 to 10: abs(12 - 10) = 2 cylinders
Move from cylinder 10 to 6: abs(10 - 6) = 4 cylinders
Move from cylinder 6 to 2: abs(6 - 2) = 4 cylinders
Total cylinders moved: 2 + 10 + 2 + 15 + 33 + 10 + 2 + 4 + 4 = 82 cylinders
Given that a seek takes 6 ms per cylinder moved, the total seek time for LOOK would be:
82 cylinders * 6 ms/cylinder = 492 ms

b A manufacturer wishes to design a hard disk with a capacity of 2^{10} GB. If the technology used to manufacture the disks allows 1 KB sector size, 4 K sectors per track, and having 2^{10} 4 platters, then how many cylinders are required in the hard disk?

Ans:

To determine the number of cylinders required for the hard disk, we first need to find the total number of sectors needed.

Given capacity: 2^{10} GB = 1024 GB

Convert GB to KB: 1024 GB × 1024 MB/GB × 1024 KB/MB = 2^{30} KB

Now, we can calculate the number of sectors needed:

Sector size: 1 KB

Total number of sectors: (2^{30} KB) / (1 KB/sector) = 2^{30} sectors

Next, we need to find the number of tracks required:

Sectors per track: 4 K sectors

Total number of tracks: (2^{30} sectors) / (4 K sectors/track) = (2^{30} sectors) / (2^{12} sectors/track) = $2^{(30-12)}$ tracks = 2^{18} tracks

We are given that there are 2^{10} platters. We can calculate the total number of tracks per platter:

Total tracks per platter: 2^{18} tracks / 2^{10} platters = $2^{(18-10)}$ tracks/platter = 2^{8} tracks/platter

Finally, we can find the number of cylinders required. Since each cylinder consists of two tracks per platter(one on the top and one on the bottom), the number of cylinders is equal to the number of tracks per platter:

Number of cylinders: 2^{8} / 2 = 128 cylinders

So, the hard disk requires 128 cylinders.

8 a Suppose that pages in a virtual address space are referenced in the order: 1 2 4 1 1 5 6 8 1 2 5 3 4 6 7 1 5. There are 3 empty frames available. Show the contents of frames after each memory reference and find the total number of page faults using "FIFO" and "LFU (Least Frequently Used)" page replacement algorithms. Also write the number of page faults for both the algorithms if the reference string is reversed. 4

Ans:

FIFO

	1	2	4	1	1	5	6	8	1	2	5	3	4	6	7	1	5
F1	1	1	1	1	1	5	5	5	1	1	1	3	3	3	7	7	7
F2		2	2	2	2	2	6	6	6	2	2	2	4	4	4	1	1
F3			4	4	4	4	4	8	8	8	5	5	5	6	6	6	5
	F	F	F	H	H	F	F	F	F	F	F	F	F	F	F	F	F

# of Page fault: 15

FIFO (Reverse Order)

	5	1	7	6	4	3	5	2	1	8	6	5	1	1	4	2	1
F1	5	5	5	6	6	6	5	5	5	8	8	8	1	1	1	1	1
F2		1	1	1	4	4	4	2	2	2	6	6	6	6	4	4	4
F3			7	7	7	3	3	3	1	1	1	5	5	5	5	2	2
	F	F	F	F	F	F	F	F	F	F	F	F	F	H	F	F	H

# of Page fault: 15

LFU

	1	2	4	1	1	5	6	8	1	2	5	3	4	6	7	1	5
F1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
F2		2	2	2	2	5	5	8	8	8	5	5	4	4	7	7	7
F3			4	4	4	4	6	6	6	2	2	3	3	6	6	6	5
	F	F	F	H	H	F	F	F	H	F	F	F	F	F	F	H	F

# of Page fault: 13

LFU (Reverse Order)

	5	1	7	6	4	3	5	2	1	8	6	5	1	1	4	2	1
F1	5	5	5	6	6	6	5	5	5	8	8	8	1	1	1	1	1
F2		1	1	1	4	4	4	2	2	2	6	6	6	6	4	4	4
F3			7	7	7	3	3	3	1	1	1	5	5	5	5	2	2
	F	F	F	F	F	F	F	F	F	F	F	F	F	H	F	F	H

# of Page fault: 15

b

4

Ans:
Each potassium thread waits to be grouped with one other potassium and oxygen before returning
Each oxygen thread waits for two other potassium before returning
Only one thread access shared state at a time

There is only one condition any thread will wait for, i.e. a Potassium Oxide molecule being formed. However, it will be necessary to signal potassium and oxygen threads independently, so we will use two condition variables, waitingP and waitingO.

It will be necessary to know the number of potassium and oxygen threads in the monitor. But it would be more useful to know how many potassium and oxygen threads have been assigned and have not been assigned to Potassium Oxide molecules; let these be int wP (number of waiting potassium), wO (number of waiting oxygen), aP (number of assigned potassium), and aO (number of assigned oxygen). These are all initialized to 0.

```
Potassium() {
        lock.acquire();
        wP++;
        while(aP==0) {
                if(wP>=2 && wO>=1) {
                        wP-=2; aP+=2;
                        wO-=1; aO+=1;
                        waitingP.signal();
                        waitingO.signal();
                }
                else {
                        waitingP.wait();
                }
        }
        aP--;
        lock.release();
}

Oxygen() {
        lock.acquire();
        wO++;
        while(a==0) {
                if(wP>=2 && wO>=1) {
                        wP-=2; aP+=2;
                        wO-=1; aO+=1;
                        waitingP.signal();
                        waitingP.signal();
                }
                else {
                        waitingO.wait();
                }
        }
        aO--;
        lock.release();
```

Partly correct, part marks can be given.

*****