SET D

SRM Institute of Science and Technology
College of Engineering and Technology
School of Computing

SRM Nagar, Kattankulathur – 603203, Chengalpattu District, Tamilnadu

Academic Year: 2023-24 (ODD)

Test: FJ2 | **Date: 01.10.2024**

Course Code & Title: 21CSC202J - Operating Systems | **Duration:** 100 Minutes

Year & Sem: II Year / III Sem | **Max. Marks:** 50

Course Articulation Matrix: ***(to be placed)***

S.No	Course Outcome	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
1	CO1	3	3	2	2								3			
2	CO2	3	3	3	2								3			
3	CO3	3	3	3	2								3			
4	CO4	3	3	3	2								3			
5	CO5	3	2	3	2								3			

Questions with Answer Key

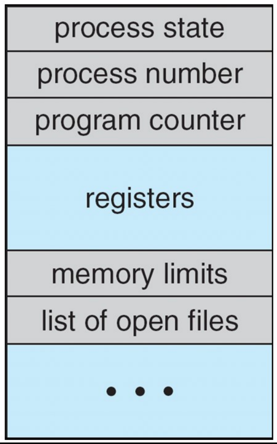
Part – A
(10 x 1 = 10 Marks)

Instructions: Answer all

Q. No	Answer with choice variable	Marks	BL	CO	PO	PI Code
1	The address of the next instruction to be executed by the current process is provided by the ___________ a) CPU registers b) Program counter c) Process stack d) Pipe Answer: b	1	L1	2	2	1.2.2
2	Which of the following is also called job scheduler? a. DMA controller b. CPU Scheduller c. Short Term Scheduller d. Long Term Scheduller Answer : d	1	L1	2	2	1.6.1

3	Each process identified and managed via a ------------- - a.Process Control Block b.Device Queue c.Process Identifier d.schedulling ID. Answer : c	**1**	**L2**	**2**	**3**	**1.5.1**
4	Mutual exclusion can be provided by the __________ a) mutex locks b) binary semaphores c) both mutex locks and binary semaphores d) none of the mentioned Answer: c	**1**	**L1**	**2**	**2**	**1.6.1**
5	Multiple user-level threads are mapped to a single kernel thread is called as ___________ a) many-to-one thread b) one to one thread c) many to many d) two level thread Answer: a	**1**	**L2**	**2**	**2**	**1.6.1**
6	In the deadlock system model, what is meant by "mutual exclusion"? A) Only one process can use a resource at a time. B) Processes hold resources while waiting for additional resources. C) Resources cannot be forcibly taken from processes holding them. D) A circular chain of processes exists, each holding a resource the next process needs	**1**	**1**	**3**	**1**	**1.3.1**

	Answer: A					
7	For a deadlock to arise, which of the following conditions must hold simultaneously? a) Mutual exclusion b) No pre-emption c) Hold and wait d) All of the mentioned Answer : D	**1**	1	3	1	1.2.1
8	For Mutual exclusion to prevail in the system ____________ a) at least one resource must be held in a non sharable mode b) the processor must be a uniprocessor rather than a multiprocessor c) there must be at least one resource in a sharable mode d) all of the mentioned Answer : A	**1**	1	3	1	1.3.1
9	The processes that are residing in main memory and are ready and waiting to execute are kept on a list called _____________ a) job queue b) ready queue c) execution queue d) process queue Answer : b) ready queue	**1**	**2**	**3**	**2**	**2.2.3**
10	The interval from the time of submission of a process to the time of completion is termed as ____________ a) waiting time b) turnaround time c) response time d) throughput Answer: b) turnaround time	**1**	**2**	**3**	**2**	**2.2.3**
Part – B (4 x 5 = 20 Marks)						
11	**Write the components in the Process Control Block with diagram.** **Answer:** Information associated with each process (also called task control block) • Process state – running, waiting, etc • Program counter – location of instruction to next execute • CPU registers – contents of all process-centric registers • CPU scheduling information- priorities, scheduling queue pointers • Memory-management information – memory allocated to the process • Accounting information – CPU used, clock	**4**	**L2**	**2**	**2**	**2.6.2**

	time elapsed since start, time limits • I/O status information – I/O devices allocated to process, list of open files  					

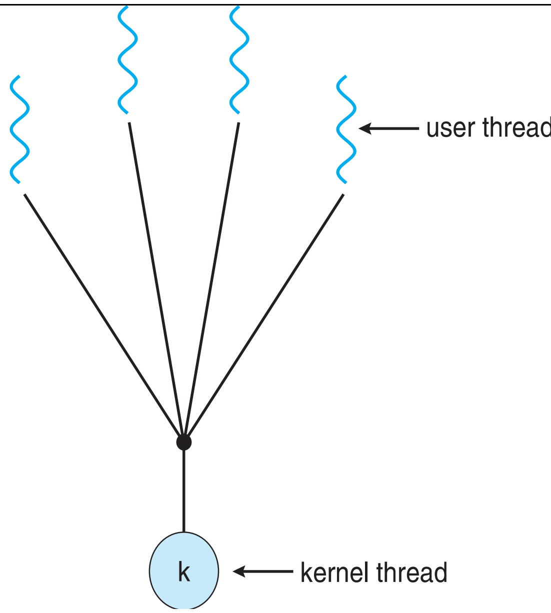
12	**Describe the various hardware solutions to critical section problems.**
	Answer :
	Compare and swap
	`do {`
	`while (compare_and_swap(&lock, 0,`
	`; /* do nothing */`
	`/* critical section */`
	`lock = 0;`
	`/* remainder section */`
	Test and set
	`do {`
	`while (test_and_set(&lock)`
	`; /* do nothing */`
	`/* critical sectior`
	`lock = false;`
	`/* remainder sectic`
		4	**L3**	**2**	**3**	**2.6.4**

13	Compare and contrast preemptive and non-preemptive scheduling algorithms. Ans : Preemptive Scheduling: This type of scheduling allows the currently running process to be interrupted and moved to the ready state, enabling another process to use the CPU. This approach is suitable for time-sharing systems where quick response times are necessary. Examples include Round Robin (RR) and Preemptive Priority Scheduling. Non-Preemptive Scheduling: Once a process starts its execution, it cannot be interrupted and will run until it completes its CPU burst. This type of scheduling is simpler and has lower overhead since there is no need to handle context switching as frequently. Examples include First-Come, First-Served (FCFS) and Shortest Job First (SJF) without preemption.	**4**	**2**	**3**	**1**	**1.3.1**

14	**Describe the deadlock recovery mechanism.**
	Answer :
	i)Process Termination
	• Abort all deadlocked processes
	• Abort one process at a time until the deadlock cycle is eliminated
	• Abort can be done in the following order
	  • Priority of the process
	  • How long process has computed, and how much longer to completion
	  • Resources the process has used
	  • Resources process needs to complete
	  • How many processes will need to be terminated
	ii) **Resource Preemption**
	  • Selecting a victim – which resource or which process to be preempted? minimize cost
	  • Rollback – return to some safe state, restart process for that state ie. Rollback the process as far as necessary to break the deadlock.
	  • Problem: starvation – same process may always be picked as victim, include number of rollback in cost factor
	  • Ensure that process can be picked as a victim only finite number of times.
		4	2	3	1	1.2.1

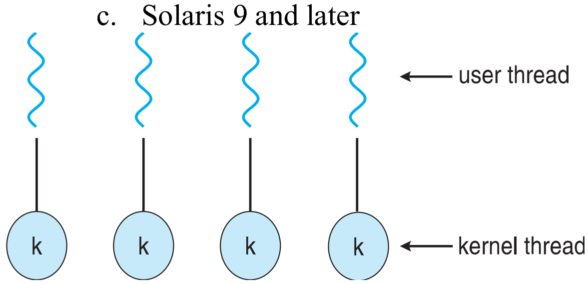
Part – C **Either OR Choice Questions** **(2 X 10 = 20 Marks)**						

15a	**Explain the various multi-threading models with necessary diagrams**
	Answer:
	i) Many-to-One
	• Many user-level threads mapped to single kernel thread
	• One thread blocking causes all to block
	• Multiple threads may not run in parallel on muticore system because only one may be in kernel at a time
	• Few systems currently use this model
	• Examples:
	  a. Solaris Green Threads
	  b. GNU Portable Threads
		10	L2	2	3	2.6.2



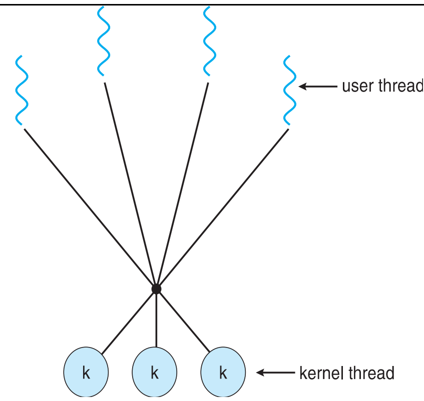
ii) One-to-One

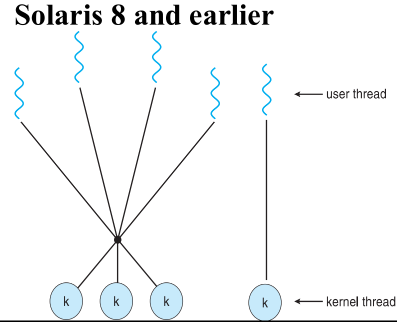
- Each user-level thread maps to kernel thread
- Creating a user-level thread creates a kernel thread
- More concurrency than many-to-one
- Number of threads per process sometimes restricted due to overhead
- Examples
    a. Windows
    b. Linux
    c. Solaris 9 and later



iii) Many-to-Many Model

- **Allows many user level threads to be mapped to many kernel threads**
- **Allows the operating system to create a sufficient number of kernel threads**
- **Solaris prior to version 9**
- **Windows with the *ThreadFiber* package**

<table>
<tr><td></td><td>

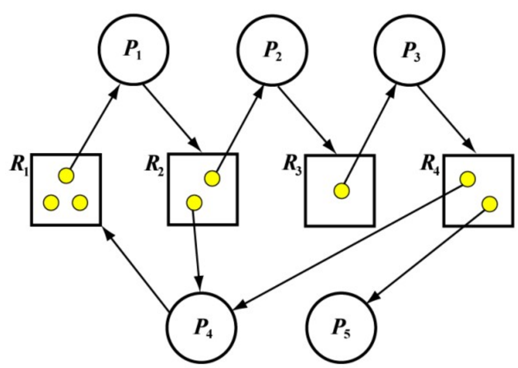
iv) Two-level Model
• Similar to M:M, except that it allows a user thread to be bound to kernel thread
• Examples
• IRIX
• HP-UX
• Tru64 UNIX
• Solaris 8 and earlier

</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="7">(or)</td></tr>
<tr><td>15b</td><td>Consider a situation of having concurrent read and write operation over a common resource like database. In which, many users wants to read and write on the same database. If many users are concurrently perform read operation, it will not create any problem. Whereas if a write operation and any other operation(may be read or write) are concurrently performed on the common field may leads to inconsistencies in the database content. Write the semaphore solution for readers-writer problem.
Answer:
• Shared Data
– Data set
– Semaphore rw_mutex initialized to 1
– Semaphore mutex initialized to 1
– Integer read_count initialized to 0</td><td>10</td><td>L3</td><td>2</td><td>3</td><td>2.6.4</td></tr>
</table>

The structure of a writer process

```
    do {
        wait(rw_mutex);
           ...
        /* writing is performed */
           ...
      signal(rw_mutex);
   } while (true);
```

The structure of a reader process

```
   do {
        wait(mutex);// lock for updating the read count variable
        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);
```

16a	Consider the following resource allocation graph.  (a) Convert it to the matrix representation (i.e., Allocation, Request and Available). (b) Do a step-by-step execution of the deadlock detection algorithm. For each step, add and remove the directed edges, and redraw the resource allocation graph. (c) Is there a deadlock? If there is a deadlock, which processes are involved? Answer:	**10**	3	3	3	3.2.3

a)

Answer: The matrix representation of the given resource allocation graph is shown below:

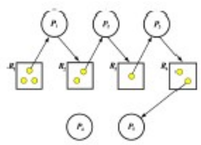
	Allocation				*Request*				*Available*			
	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4
P_1	1	0	0	0	0	1	0	0	2	0	0	0
P_2	0	1	0	0	0	0	1	0				
P_3	0	0	1	0	0	0	0	1				
P_4	0	1	0	1	1	0	0	0				
P_5	0	0	0	1	0	0	0	0				

b)

Because P_4's $Request = [1,0,0,0] \leq Available = [2,0,0,0]$, P_4 runs and returns is $Allocation = [0,1,0,1]$ making the new $Available = [2,0,0,0] + [0,1,0,1] = [2,1,0,1]$. The matrix representation becomes:

	Allocation				*Request*				*Available*			
	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4
P_1	1	0	0	0	0	1	0	0	2	1	0	1
P_2	0	1	0	0	0	0	1	0				
P_3	0	0	1	0	0	0	0	1				
P_4												
P_5	0	0	0	1	0	0	0	0				

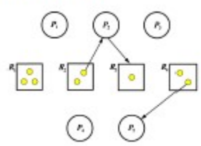
This is the corresponding resource allocation graph:



Then, we can run P_1 because P_1's $Request = [0,1,0,0] \leq Available = [2,1,0,1]$. After reclaiming P_1's $Allocation = [1,0,0,0]$, the new $Available$ is old $Avaliable = [2,1,0,1] + P_1'$s $Allocation = [1,0,0,0] = [3,1,0,1]$. The new matrix representation is:

	Allocation				*Request*				*Available*			
	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4
P_1									3	1	0	1
P_2	0	1	0	0	0	0	1	0				
P_3	0	0	1	0	0	0	0	1				
P_4												
P_5	0	0	0	1	0	0	0	0				

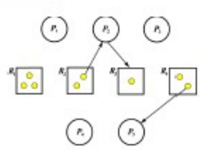
Here is the corresponding resource allocation graph:



The next process is P_3 because P_3's $Request = [0,0,0,1] \leq Available = [3,1,0,1]$. After P_3 finishes its work, its $Allocation = [0,0,1,0]$ is returned to $Available = [3,1,0,1] + [0,0,1,0] = [3,1,1,1]$:

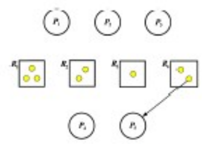
	Allocation				*Request*				*Available*			
	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4
P_1									3	1	1	1
P_2	0	1	0	0	0	0	1	0				
P_3												
P_4												
P_5	0	0	0	1	0	0	0	0				

The resource allocation graph is shown below:



Now we can run P_2 and the yields (*i.e.*, matrix representation and resource allocation graph) are:

	Allocation				*Request*				*Available*			
	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4
P_1									3	2	1	1
P_2												
P_3												
P_4												
P_5	0	0	0	1	0	0	0	0				



Finally, we can run P_5 and all processes are done! Note that there are cycles $P_1 \rightarrow R_2 \rightarrow P_4 \rightarrow R_1 \rightarrow P_1$ and $P_1 \rightarrow R_2 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_4 \rightarrow P_4 \rightarrow R_1 \rightarrow P_1$. However, there is no deadlock. ■

c)

Finally, we can run P_5 and all processes are done! Note that there are cycles $P_1 \rightarrow R_2 \rightarrow P_4 \rightarrow R_1 \rightarrow P_1$ and $P_1 \rightarrow R_2 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_4 \rightarrow P_4 \rightarrow R_1 \rightarrow P_1$. However, there is no deadlock. ■

(Or)

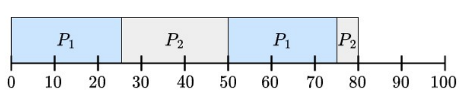
16b	Consider two processes, $P1$ and $P2$, where $p1 = 50$, $t1 = 25$, $p2 = 75$, and $t2 = 30$. a. Can these two processes be scheduled using rate-monotonic scheduling? Illustrate your answer using a Gantt chart. (5 Marks) b. Illustrate the scheduling of these two processes using earliestdeadline-first (EDF) scheduling. (5 Marks)	**10**	3	3	3	3.2.3

Answer:

Let's first understand the given data:

- P_1 and P_2 are periodic tasks;
- P_1 has a period of 50 units ($p_1 = 50$) and an execution time of 25 units ($t_1 = 25$);
- P_2 has a period of 75 units ($p_2 = 75$) and an execution time of 30 units ($t_2 = 30$).

Since P_1 has a shorter period than P_2, it has a higher priority than P_2. Thus, P_1 will be scheduled first at $t = 0$ and will run until $t = 25$. Then, P_2 will run from $t = 25$ to $t = 50$, untill the deadline of P_1. Then, P_2 will be preempted by P_1 and will run again from $t = 50$ to $t = 75$. Until this point, P_2 ran for $t = 50 - 25 = 25$ units and P_1 ran for $t = 50 + 25 = 75$ units which is the deadline of P_2, but it stills needs to complete $t = 30 - 25 = 5$ units. Hence, it can be said that the two processes cannot be scheduled using RMS. The Gantt chart for the two processes is shown in the following figure:



Course Outcome (CO) and Bloom's level (BL) Coverage in Questions