USN | | | | | | | | | | |

Internal Assessment Test 2 – January 2023

Sub:	**OPERATING SYSTEMS -Answer Key**			Sub Code:	**BCS/BAD 303**	Branch:	**AINDS / CS (DS)**
Date:	**19/01/2023**	Duration:	**90 minutes**	Max Marks: **50**	Sem	**III**	**OBE**

		Answer any FIVE Questions	MARKS	CO	RBT
1	a	**Write a note on multithreading models with neat sketches** Many to many model. Many to one model. one to one model. Explanation with diagram ——>**5 marks**	[5]	1	L2
	b	**Explain Threading Issues.** These are the **issues to be consider in multithreaded programs** ● Semantics of fork() and exec() system calls ● Thread cancellation ● Signal handling ● Thread pools ● Thread specific data ● Scheduler activations Any Five issues with explanation ——--------->**5 marks**	[5]	1	L2
2		**For the following set of processes, find the avg. waiting time and avg. turn around using Gantt chart for** **a) FCFS** **b) SJF (primitive)** **Process Arrival Time Burst Time** **P1 0 4** **P2 1 2** **P3 2 5** **P4 3 4** **a) Gantt chart** **3 Marks** | P1 | P2 | P3 | P4 |	[10]	2	L4

<table>
<tr>
<td></td>
<td></td>
<td>
<table><tr><td>0 4</td><td>6</td><td>11</td><td>15</td></tr></table>

AVG WAIT TIME= (0+4+6+11)/4= 21/4=5.25 ms 1 Mark

Avg TAT=(4+6+11+15)/4= 36/4= 9 ms 1 Mark

If they are considering AT then
Avg wait time=(0+3+4+8)/4= 3.75ms
Avg TAT=(4+5+9+12)/4= 7.5ms

b) Gantt chart 3 Marks
<table><tr><td>P1</td><td>P2</td><td>P1</td><td>P4</td><td>P3</td></tr><tr><td>0 1</td><td>3</td><td>6</td><td>10</td><td>15</td></tr></table>

Avg Wait Time=(2+0+8+3)/4= 13/4= 3.25 ms 1 Mark

Avg TAT=(6+2+13+7)/4= 28/4 = 7 ms 1 Mark
</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>
What are semaphores? Explain two primitive semaphore operations. What are its advantages?
It is a synchronization tool that is used to generalize the solution to the critical section problem in complex situations. --------->1mark
A Semaphore s is an integer variable that can only be accessed via two indivisible (atomic) operations namely
1. wait or P operation (to test)
2. signal or V operation (to increment) -------->1 Marks
</td>
<td></td>
<td>3</td>
<td>L2</td>
</tr>
</table>

definition of wait() is as follows:

```
wait(S) {
    while S <= 0
      ; // no-op
    S--;
}
```

The definition of signal() is as follows:

```
signal(S) {
    S++;
}
```

--------> 2 marks

Signal and Wait with explanation

Counting Semaphore —---------->**2 Marks**

Binary semaphore —-------->**2 Marks**

Advanatges: **--------->2 Marks**

- Semaphores allow only one process into the critical section. They follow the mutual exclusion principle strictly and are much more efficient than some other methods of synchronization.
- There is no resource wastage because of busy waiting in semaphores as processor time is not wasted unnecessarily to check if a condition is fulfilled to allow a process to access the critical section.

4 | a | [8] | 3 | L4

Considering a system with five processes P0 through P4 and three resources of type A, B, C. Resource type A has 10 instances, B has 5 instances and type C has 7 instances. Suppose at time t0 following snapshot of the system has been taken:

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	

What will be the content of the Need matrix?

Is the system in a safe state? If yes, then what is the safe sequence?

What will happen if process P1 requests one additional instance of resource type A and two instances of resource type C?

Process	Allocation			Max			Available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2	7	4	3
P1	2	0	0	3	2	2				1	2	2
P2	3	0	2	9	0	2				6	0	0
P3	2	1	1	2	2	2				0	1	1
P4	0	0	2	4	3	3				4	3	1

Need Matrix —-----> **2 Marks**
Safe State —-->**1 mark**
Safe Sequence: P1, P3, P4, P0, P2 —----> **1 Mark**

P1 with additional request

Total A=10, B=5, C=7

Process	Allocation			Max			Available			Need		
	A	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
P0	0	1	0	7	5	3	3	3	2	7	4	3
P1	2	0	0	4	2	5	5	4	3	2	2	5
P2	3	0	2	9	0	2	5	4	5	6	0	0
P3	2	1	1	2	2	2	7	4	5	0	1	1
P4	0	0	2	4	3	3	10	4	7	4	3	1

Need Matrix —-------> **2 Marks**
Safe sequence: P3, P4, P1, P2,P0 —-------> **1 Mark**
System is in Safe State —-------> **1 Mark**

b

What are the three essential requirements that a solution to the critical-section problem must satisfy?

—------> **2 Marks**

1. **Mutual Exclusion** - If process **P_i** 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 there exist some processes that **wish to enter their critical section**, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely.

3. **Bounded Waiting** - A bound must exist 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.

[2] 3 L2

5 a

State dining philosophers problem and give a solution using

[5] 3 L3

semaphores

- The dining philosophers problem is a **classic synchronization problem** involving the allocation of limited resources among a group of processes in a **deadlock-free and starvation-free manner:**

- Consider five philosophers sitting around a table, in which there are five chopsticks evenly distributed and an endless bowl of rice in the center, as shown in the diagram below. (There is exactly one chopstick between each pair of dining philosophers.)
    - These philosophers spend their lives alternating between two activities: **eating and thinking.**
    - When it is time for a philosopher to eat, **it must first acquire two chopsticks** - one from their left and one from their right.
    - When a philosopher thinks, **it puts down both chopsticks** in their original locations.

——----->**2 Marks**

Solution using semaphore ——-------**3 Marks**

The structure of Philosopher *i*:

```
do {
        wait ( chopstick[i] );  // left chopstick
        wait ( chopStick[ (i + 1) % 5] );  //right chopstick
                                                   <- Entry Section
           Critical Section    //  eat

        signal ( chopstick[i] );   //release left chopstick
        signal (chopstick[ (i + 1) % 5] );  // release right chopstick
                                                   <- Exit Section

           Remainder Section    //  think

} while (TRUE);
```

	b	**Define Deadlock and mention necessary conditions for deadlock. Explain resource allocation graph with diagram.**	[5]	3	L2

<table>
<tr>
<td></td>
<td></td>
<td>
The waiting process is never again able to change state because the resource it has requested are held by other waiting processes. This situation is called a deadlock —---> 1 Mark

- Mutual exclusion:
- Hold and wait:
- No Preemption
- Circular Wait —------>2 Marks

Explaining Resource allocation graph with example .---------> 2 Marks

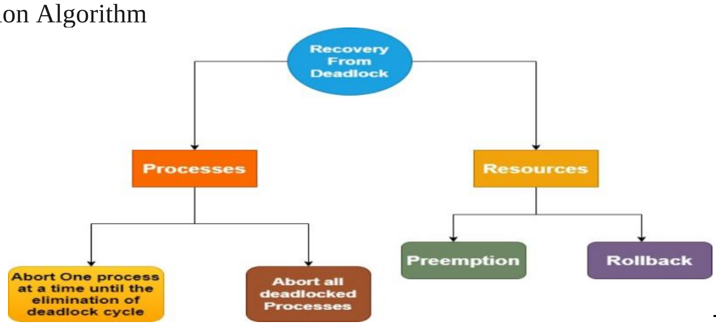
- V is partitioned into two types:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the processes in the system.
  - $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system.
- request edge – directed edge $P_1 \rightarrow R_j$
- assignment edge – directed edge $R_j \rightarrow P_i$
</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>
Explain about Deadlock Detection and explain various methods for recovering from a deadlock?

1. Single instance —------> 3 Marks

If all the resources have only a single instance, then a deadlock-detection algorithm can be defined that mainly uses the variant of the resource-allocation graph and is known as a wait-for graph. This wait-for graph is obtained from the resource-allocation graph by removing its resource nodes and collapsing its appropriate edges.

2. Multiple Instance —-------->3 Marks

Detection Algorithm



------->4 Marks
</td>
<td>[10]</td>
<td>3</td>
<td>L2</td>
</tr>
</table>