Type: MCQ

Q1. Consider a set of n tasks with known runtimes r1, r2, ..., rn to be run on a uniprocessor machine., Among the following options, select the CPU scheduling algorithm that will result in the maximum throughput. (1)

1. Round-Robin
2. **Shortest -Job -First
3. First -Come-First-Served
4. Best-Fit

Q2. A counting semaphore was initialized to 2. Then 5 P (wait) operations and 3 V (signal) operations were completed on this semaphore. Select the resulting value of the semaphore. (1)

1. 8
2. 6
3. ** 0
4. 10

Q3. Consider the following code snippet using the fork( ) and wait ( ) system calls. Assume that the code compiles and runs correctly, and that the system calls run successfully without any errors.

```
int x=2;
while(x>0)
{
 fork ( );
 printf("hello\n");
 wait(NULL);
 x--;
}
```

Select the total number of times the printf( ) statement executed. (1)

1. 8
2. 12
3. **6
4. 10

Q4. Choose the correct options that are used by the system to ensure that the system will never enter a deadlock state.

a. deadlock prevention

b. deadlock detection

c. deadlock recovery

d. deadlock avoidance (1)

1. a and b
2. b and c
3. c and d
4. ** d and a

Q5.

Process Pi	Process Pj
Entry Section: loop while(____);	Entry Section: loop while(____);
(Critical Section)	(Critical Section)
Exit Section: turn=1;	Exit Section: turn=0;

Consider the above two processes Pi and Pj. Suppose turn is a variable whose value can be either 0 or 1. Let's assume, turn is initialized to 0. Choose the code for the blank spaces inside the while loops, so that both processes follow mutual exclusion and bounded waiting property. (1)

1. turn==0 in process Pi and turn==0 in process Pj
2. **turn==1 in process Pi and turn==0 in process Pj
3. turn==0 in process Pi and turn==1 in process Pj
4. turn==1 in process Pi and turn==1 in process Pj

Q6. Select the false statement (1)

1. **Multiprogramming will not cause starvation
2. Timesharing improves performance
3. Kernel runs in protected mode
4. Scheduler can prioritize some applications

Q7. Select the true statement related to preemptive scheduling algorithm (1)

1. CPU is allocated for the entire burst time to a process
2. **A low priority process will wait if a high priority process arrives simultaneously
3. Race conditions will never occur if data are shared among multiple processes
4. All statements are correct

Q8. Select the true statement related to inter process communication (1)

1. Messages are waiting in a zero-capacity buffer
2. The sender is blocked for the unbounded capacity buffer if it is full
3. ** Messages are stored in a temporary buffer
4. All statements are correct

Q9. Select the process state transition that occurs due to the scanf() system call in the following program is

```
int main()
{
  int a;
  scanf("%d\n",&a);
  exit(0);
}    (1)
```

1. ** Running -> Waiting
2. New -> Ready
3. Ready -> Running
4. Waiting -> Ready

Q10. Select the true statement (1)

1. Thread communication is easier than inter-process communication
2. Multiple Threads share common address space
3. Threads increases the throughput of the system
4. **All options are correct

Type: DES

Q11 A. Consider the following set of processes, with the length of CPU bursts in milliseconds.

Process	P1	P2	P3	P4	P5
Arrival Time	00	02	03	06	30
Burst	10	12	14	16	05

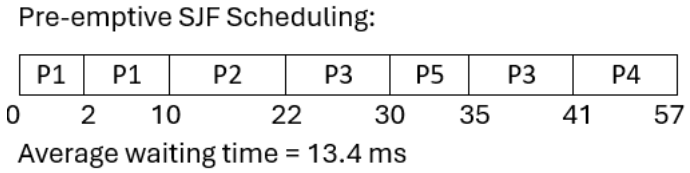
I. Apply preemptive Shortest Job First scheduling algorithm on the given set of processes. With the help of a Gantt chart, illustrate the execution of these processes. Calculate the average waiting time.

II. Apply preemptive Priority scheduling algorithm on the given set of processes. Given priority of each process is P1=4, P2=3, P3=5, P4=1, P5=1. Smaller numbers represent higher priority. With the help of a Gantt chart, illustrate the execution of these processes. Calculate the average waiting time. (5)

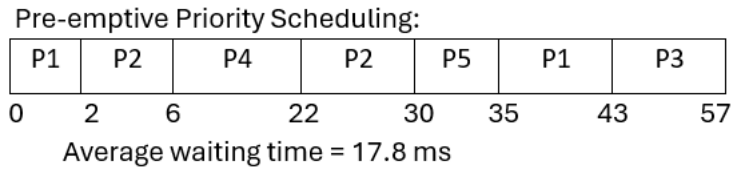
Solution:

I.

Correct illustration of Gnatt Chart : 1.5 Points, Average Waiting Time: 1 Point



II. **Correct illustration of Gnatt Chart : 1.5 Points, Average Waiting Time: 1 Point**



Q11 B. Analyse how race conditions have occurred in producer-consumer problems with a suitable example. (3)

Solution: Analysis (1) + suitable example (2)

Race condition is a situation where several processes access and manipulate the same data concurrently and the outcome of the execution depends on the order in which the access takes place.

Producers and consumers are co-operating processes. A producer process produces information that is consumed by a consumer process. Buffer is used as a shared memory between these two processes. Let's assume, *counter* is the size of the buffer. Regarding this, when a producer process produces a data, *counter* will be incremented by 1 (counter++). On the other hand, when a consumer process consumes the data, *counter* will be decremented by 1 (counter--). Here, the producer and the consumer process both use a shared variable *counter*. One example of race conditions happening in producer-consumer problem, if counter++ and counter—are executed in the following sequence.

counter++;

counter--;

counter++ and counter-- are implemented in machine language in the following way.

• counter++ could be implemented as

register1 = counter

register1 = register1 + 1

counter = register1

• counter-- could be implemented as

register2 = counter

register2 = register2 - 1

counter = register2

• Consider this execution interleaving with “counter = 5” initially:

S0: producer execute register1 = counter {register1 = 5}

S1: producer execute register1 = register1 + 1 {register1 = 6}

S2: consumer execute register2 = counter {register2 = 5}

S3: consumer execute register2 = register2 – 1 {register2 = 4}

S4: producer execute counter = register1 {counter = 6}

S5: consumer execute counter = register2 {counter = 4}

Hence, the final value of counter is 4, which is incorrect. It must be 5. This example of race conditions results in data inconsistency.

Q11 C. Explain different kinds of process states and state transition using a neat diagram. (2)

Solution: Explanation (1) + Neat Diagram (1)

As a process executes, it changes state. Different sets are as follows

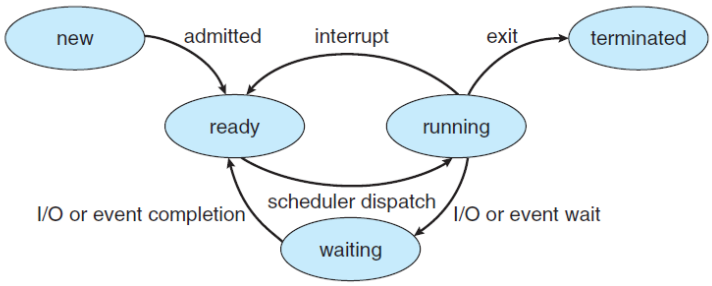
new: The process is being created

running: Instructions are being executed

waiting: The process is waiting for some event to occur

ready: The process is waiting to be assigned to a processor

terminated: The process has finished execution



Q12A. Consider the following snapshot of a system.

	Allocation	Max	Available
Processes	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	

Apply Banker's algorithm to check whether the system is in safe state. If so, give the safe sequence. If not, justify your answer. Show all the steps clearly. (5)

Solution: Application of Banker's algorithm with clear specification of all the steps (4 Point) + Safe Sequence (1 Point)

Steps:

1. Calculation of Need = Max - Allocation

Hence,

Process	**Allocation**				**Max**				**Work or Available**				**Need**			
	A	**B**	**C**	**D**	**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	2	2	1	1
P1	3	1	2	1	5	2	5	2					2	1	3	1
P2	2	1	0	3	2	3	1	6					0	2	1	3
P3	1	3	1	2	1	4	2	4					0	1	1	2
P4	1	4	3	2	3	6	6	5					2	2	3	3

2. Finish[P0] = false; Finish[P1] = false; Finish[P2] = false; Finish[P3] = false; Finish[P4]= false;

3.

(i) Is $Need_{P0}$ <= Work or Available? Yes, <2,2,1,1> <= <3,3,2,1>

Safe Sequence:<P0>

Allocation of P0 is <2, 0, 0, 1>. After completion of P0,

Work or Available = <3, 3, 2, 1> + <2, 0, 0, 1> = <5, 3, 2, 2> and Finish[P0] = true;

(ii) Is $Need_{P1}$ <= Work or Available?

No, <2,1,**3**,1>= <5,3,**2**,2> ; In this moment, P1 cannot be added towards the safe sequence

(iii) Is $Need_{P2}$ <= Work or Available?

No, <0,2,1,**3**> <= <5,3,2,**2**>; In this moment, P2 cannot be added towards the safe sequence

(iv) Is $Need_{P3}$ <= Work or Available? Yes, <0,1,1,2> <= <5,3,2,2>

Safe Sequence:<P0, P3>

Allocation of P3 is <1, 3, 1, 2>. After completion of P3,

Work or Available = <5, 3, 2, 2> + <1, 3, 1, 2> = <6, 6, 3, 4> and Finish[P3] = true;

(v) Is $Need_{P4}$ <= Work or Available? Yes, <2,2,3,3> <= <6, 6, 3, 4>

Safe Sequence:<P0, P3, P4>

Allocation of P4 is <1, 4, 3, 2>. After completion of P4,

Work or Available = <6, 6, 3, 4> + <1, 4, 3, 2> = <7, 10, 6, 6> and Finish[P4] = true;

(vi) Is $Need_{P1}$ <= Work or Available? Yes, <2,1,3,1> <= <7, 10, 6, 6>

Safe Sequence:<P0, P3, P4, P1>

Allocation of P1 is <3, 1, 2, 1>. After completion of P1,

Work or Available = <7, 10, 6, 6> + <3, 1, 2, 1> = <10, 11, 8, 7> and Finish[P1] = true;

(vii) Is $Need_{P2}$ <= Work or Available? Yes, <0,2,1,3> <= <10, 11, 8, 7>

Safe Sequence:<P0, P3, P4, P1, P2>

Allocation of P2 is <2, 1, 0, 3>. After completion of P2,

Work or Available = <10, 11, 8, 7> + <2, 1, 0, 3> = <12, 12, 8, 10> and Finish[P2] = true;

4.

Since, we get Finish[i] = true for all processes, the system is in a safe state.

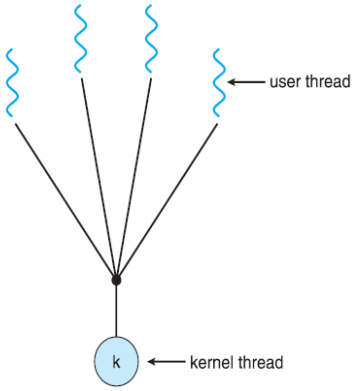
Safe sequence of the processes is P0–>P3–>P4–>P1–>P2.

Q12B. Analyze different multithreading models with neat diagrams. (3)

Solution: Analysis (2) + Diagrams (1) (Total Four types of models)

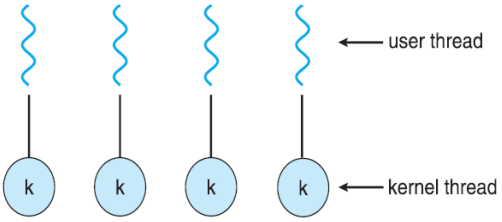
Many to one Model :

- 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: Solaris Green Threads, GNU Portable Threads



One to One Model :

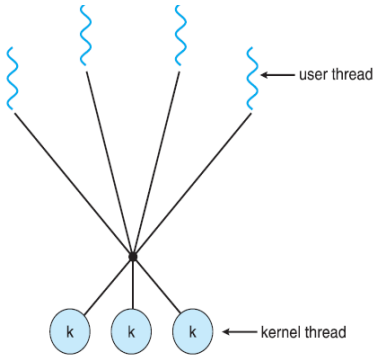
- Each user-level thread maps to kernel thread
- Creating a user-level thread creates a kernel thread
- More concurrency than many-to-one



- Number of threads per process sometimes restricted due to overhead
- Examples : Windows, Linux, Solaris 9 and later

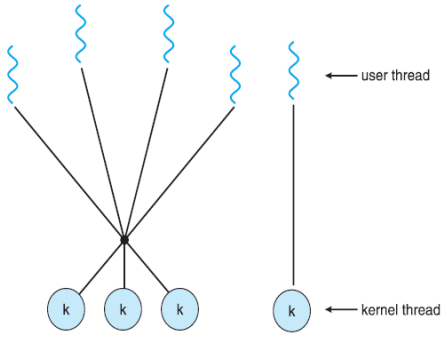
Many to Many Model:

- 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



Two level Model :

- Similar to M:M, except that it allows a user thread to be boundto kernel thread
- Examples: IRIX, HP-UX, Tru64 UNIX, Solaris 8 and earlier



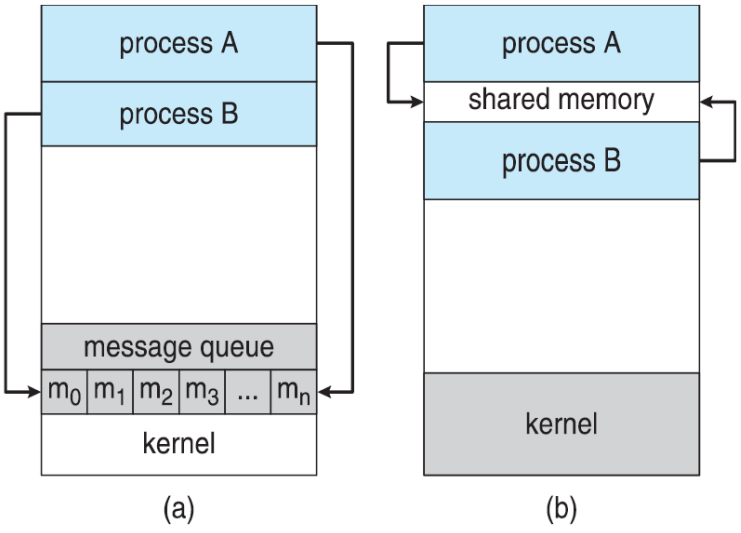
Q12C. Compare shared memory model and message-passing model in inter process communication (2)

Solution: **Comparison – 2 Points**

Inter-Process Communication – Shared Memory

- An area of memory shared among the processes that wish to communicate
- The communication is under the control of the users processes not the operating system.
- Major issues are to provide mechanism that will allow the user processes to synchronize their actions when they access shared memory

(a) Message passing. (b) shared memory.



Inter-Process Communication – 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