Question 7 [2 Points]

Part A [1 Point, 0.25 each] Suppose two threads execute the following code segment concurrently, accessing shared variables a, b, and c:

```
// Variables shared by both threads:
int a = 4;
int b = 0;
int c = 0;
```

```
//Thread 1:
if (a < 0) {
   b = 10;
   c = b - a;
   a = -3; }
else
   c = b + a;
```

```
//Thread 2:
b = 10;
a = -3;
```

What are the possible values for c after both threads complete? You can assume that reads and writes of the variables are atomic, and that the order of statements within each thread is preserved in the code generated by the C compiler. (Choose all that apply)

- 4
- 7
- 13
- 14

Part B [1 Point, 0.25 each] Consider the following processes P1, P2, P3 and P4:

P1	P2	P3	P4

`x = 55;`
`y = 4 + x;`
	`z = -1;`
	`sleep(1);`
		`z = 4;`
		`if (z > 0)`
		`  x = 20;`
			`print (y);`

Which of the following pair of processes necessitates mutual exclusion: (Choose all that apply)

- P1 and P2
- P2 and P3
- P1 and P3
- P3 and P4

Part C [1 Point, 0.5 each] Consider the following code segment for solving the critical section problem:

```
boolean intendToEnter0=false ;
boolean intendToEnter1=false;
int x = 4 ;
```

//Process P0:		//Process P1:	
while (true) {		while (true) {	
while(intendToEnter1) ;	(1)	while(intendToEnter0) ;	(1)
intendToEnter0 = true;	(2)	intendToEnter1 = true;	(2)
x = x + 5;	(3)	x = x * x;	(3)
intendToEnter0 = false;	(4)	intendToEnter1 = false;	(4)
sum= sum + (sum * 0.15)	(5)	print ("Done");	(5)
}		}	

Answer the following questions with regard to **process P0**:

1. Which line number(s) specifies the Entry Section?
   - Line (1)
   - Line (2)
   - Line (4)
   - Lines (1) and (2)

2. Which line number(s) specifies the Exit Section?
   - Line (1)
   - Line (2)
   - Line (4)
   - Lines (4) and (5)

Question 1 [3 Points, 0.5 each] Choose only ONE answer.

1. The best CPU- scheduling algorithm is the one that:
   - Maximize CPU utilization, throughput and waiting time.
   - Minimize CPU utilization, throughput and waiting time.
   - Maximize CPU utilization and minimize throughput and waiting time.
   - d) Maximize CPU utilization and throughput and minimize waiting time.

2. One solution to the starvation problem in priority scheduling is aging. Aging involves:
   -) Increasing the priority of processes that wait on the ready queue for a long time.
   -) Decreasing the priority of higher-priority processes so that lower-priority processes get a chance to run.
   - Exchanging the priorities of higher and lower priority processes.
   - None of the above.

3. Which of the following items does NOT belong to the function of a dispatcher:
   -) Switching context from one process to another.
   - Switching to user mode.
   - Selecting a process among the available ones in the ready queue.
   -) Jumping to the proper location in the user program to resume that program.

4. Which of the following statement(s) is(are) true for Multilevel feedback Queues scheduling algorithm:
   - Each queue has its own scheduling algorithm and have a higher priority over the lower priority queue.
   - Processes are partitioned according to the characteristics of their CPU bursts.
   - Each process may migrate between the queues.
   - All of the above.

5. A process is at priority 200 at a given point of time. What will be its priority after 10 minutes using an aging system in which the priority is incremented periodically every 5 seconds:
   - Zero.
   - 190.
   - 80.
   -) None of the above.

6. Which of the following statement is true about the convoy effect?
   - Round-Robin Scheduling Algorithm causes the convoy effect.
   - Occurs when there is an unbalanced mix of CPU-bound and I/O bound processes in the Ready Queue.
   - The convoy effect is a preferred phenomenon in computer systems.
   - None of the above.

Question 2 [3 Points, 0.5 each] Consider the following set of processes:

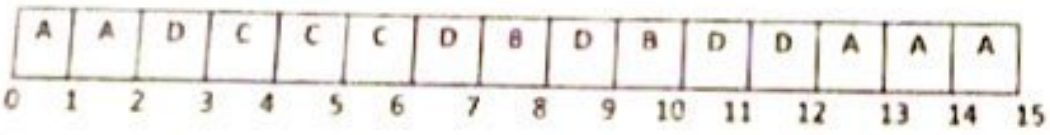
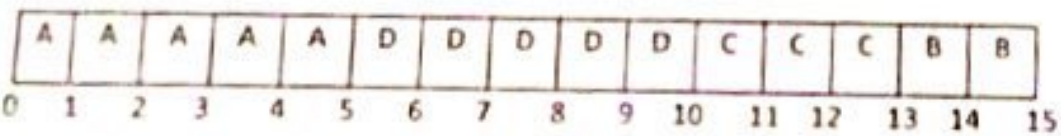
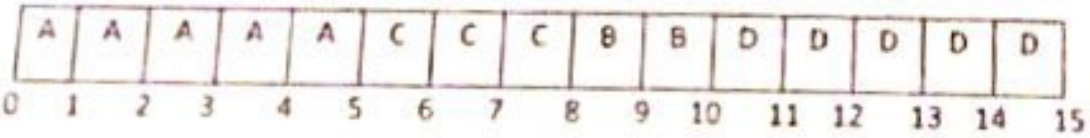
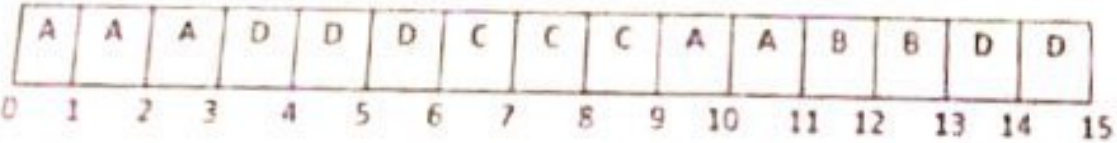
Process#	Arrival Time	Burst Time (ms)	Priority
A	0	5	4
B	6	2	2
C	3	3	1
D	2	5	2

*: *Lower number means higher priority*

Suppose the following Gantt charts show the order in which processes are executed according t specific scheduling algorithm. Match each of the following scheduling algorithms with th corresponding Gantt chart:

A. First-Come-First-Served.	**B.** (Non-preemptive) Shortest Job First.
C. (Preemptive) Shortest Remaining Time First.	**D.** Non-preemptive priority.
E. Round-Robin with time quantum 3 ms.	**F.** Round-Robin with time quantum 1ms.
G. Preemptive priority/ Round-Robin with time quantum 1ms	

Note: *When a process finishes at the same time another process arrives, the new process will be submitted to the ready queue first.*

#	Gantt Chart	Scheduling Algorithm
1		
2		
		
		

Question 8 [3 Points]

Part A [1 Point] Consider the following code segment for solving the critical section problem:

```
boolean blocked[2];
blocked[0] = false;
blocked[1] = false;
int turn = 0 ;
```

```
//Process P0:
while (true) {
blocked[0] = true;
while (turn != 0) {
while (blocked[1])
;
turn = 0; }
/*  Critical Section  */
blocked[0] = false;
/*  Remainder section */
}
```

```
//Process P1:
while (true) {
blocked[1] = true;
while (turn != 1) {
while (blocked[0])
;
turn = 1; }
/*  Critical Section  */
blocked[1] = false;
/*  Remainder section */
}
```

Does the above solution satisfy the mutual exclusion requirement? (Choose only ONE answer).

- Yes
- No

Part B [1 Point] If the programmer rewrites the above codes for processes P0 and P1 as below:

```
boolean blocked[2];
blocked[0] = false;
blocked[1] = false;
int turn = 0 ;
```

```
//Process P0:
while (true) {
blocked[0] = true;
while (turn != 0) {
while (blocked[1])
turn = 0; }
/*  Critical Section  */
blocked[0] = false;
/*  Remainder section */
}
```

```
//Process P1:
while (true) {
blocked[1] = true;
while (turn != 1) {
while (blocked[0])
turn = 1; }
/*  Critical Section  */
blocked[1] = false;
/*  Remainder section */
}
```

Does the new algorithm may cause a deadlock? (Choose only ONE answer).

- Yes
- No

Question 8 [3 Points]

Part A [1 Point] Consider the following code segment for solving the critical section problem:

```
boolean blocked[2];
blocked[0] = false;
blocked[1] = false;
int turn = 0 ;
```

```
//Process P0:
while (true) {
blocked[0] = true;
while (turn != 0) {
while (blocked[1])
;
turn = 0; }
/*  Critical Section  */
blocked[0] = false;
/*  Remainder section */
}
```

```
//Process P1:
while (true) {
blocked[1] = true;
while (turn != 1) {
while (blocked[0])
;
turn = 1; }
/*  Critical Section  */
blocked[1] = false;
/*  Remainder section */
}
```

Does the above solution satisfy the mutual exclusion requirement? (Choose only ONE answer).

- Yes
- No

Part B [1 Point] If the programmer rewrites the above codes for processes P0 and P1 as below:

```
boolean blocked[2];
blocked[0] = false;
blocked[1] = false;
int turn = 0 ;
```

```
//Process P0:
while (true) {
blocked[0] = true;
while (turn != 0) {
while (blocked[1])
turn = 0; }
/*  Critical Section  */
blocked[0] = false;
/*  Remainder section */
}
```

```
//Process P1:
while (true) {
blocked[1] = true;
while (turn != 1) {
while (blocked[0])
turn = 1; }
/*  Critical Section  */
blocked[1] = false;
/*  Remainder section */
}
```

Does the new algorithm [illegible]ause a deadlock? (Choose only ONE answer).

- Yes
- No

Question 6 [3 Points, 0.5 each] Answer the following questions. Choose only ONE answer for each question.

1. Which of the following indicates that P_j can enter the critical section in Peterson's solution?
   - flag[i] == true or turn == j
   - flag[i] == false or turn == j
   - flag[i] == false or turn == i
   - flag[i] == true and turn == i

2. When mutex lock is implemented as a binary semaphore, what should its value be initialized to be?
   - 0
   - 1
   - -1
   - None of the above

3. Which of the following may cause a liveness failure?
   - An infinite loop.
   - A busy waiting loop.
   - A deadlock.
   - All of the above.

4. A counting semaphore was initialized to 8. If we assumed that 5 *wait operations* followed by 2 *signal operations* were completed on this semaphore. Then, the resulting value of the semaphore is:
   - 5
   - 8
   - 11
   - 15

5. What is the correct order of operations for protecting a critical section using a binary semaphore?
   - `block()` followed by `wakeup()`
   - `acquire()` followed by `release()`
   - `wait()` followed by `signal()`
   - `acquire()` followed by `signal()`

6. A situation in which two or more processes are waiting indefinitely for an event that can be caused only by one of the waiting processes is called:
   - Mutex Locks.
   - Deadlocked.
   - Spinlocks.
   - None of the above.

a) Calculate the waiting time of process A?

b) Calculate the turnaround time of process E?

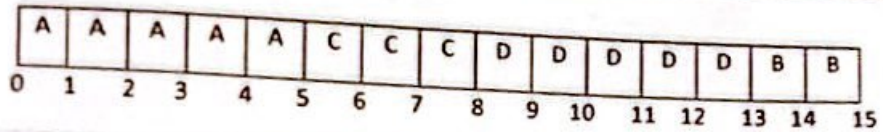
c) Calculate the response time of process C?

d) Calculate the CPU utilization rate (in percentage)?

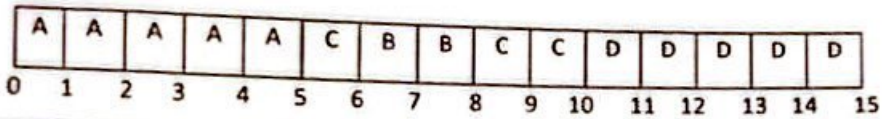
Question 5 [2 Points, 0.25 each] Indicate whether each statement is True **(T)** or False **(F)**.

Statement	T or F
1. A preemptive kernel may be more responsive than non-preemptive kernel.	
2. Concurrent or parallel execution of processes can contribute to issues involving the integrity of data shared by several processes.	
3. A counting semaphore is functionally equivalent to a mutex locks.	
4. Peterson's solution can be used to synchronize a set of processes that alternate execution between their critical sections and remainder sections.	
5. The main disadvantage of Spinlocks is that it requires busy waiting.	
6. Bounded waiting ensures that programs will cooperatively determine what process will next enter its critical section.	
7. Race condition is a situation in which multiple threads read or write a shared data item and the final value of the data depends on the last process that reads the shared data.	
8. If a process performs a signal operation on a counting semaphore whose value is zero, and there are one or more other processes waiting on the semaphore, exactly one of the other processes will be unblocked.	

5



6



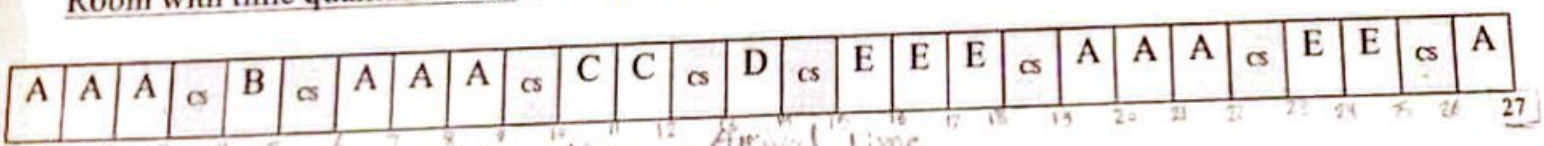
Question 3 [2 Points, 0.5 each] Indicate whether each statement is True (T) or False (F).

Statement	T or F
1. In Round-Robin scheduling, the time quantum should be small with respect to the context-switch time.	
2. In symmetric multiprocessing, all system activities are handled by a single processor – the master server.	
3. Round-Robin scheduling algorithm is suitable for a multi-user environment.	
4. A preemptive scheduling can take place when a process switches from Running state to Waiting state.	

Question 4 [2 Points, 0.5 each] Consider the following set of processes, with the length of the CPU burst given in milliseconds.

Process	Arrival Time	Burst Time (ms)
A	0	10
B	1	1
C	4	2
D	5	1
E	5	5

The following Gantt chart shows the order in which these processes are executed based on a Round Robin with time quantum=3 ms. Assume the overhead of Context Switching (CS) is 1 ms:



Turnaround time A = Completion time − Arrival time

27 − 0 = 27