Computer Science & Engineering
IIT Kharagpur
CS31202 Operating System
3rd year CSE: 6th Semester (Class Test 1)
Time limit: 1hour 15 minutes, Date: 2nd Feb, 2026
Max Marks: 20

Roll No: _____

Name: _____

1. A shared variable `Balance` represents the balance of a bank account, and is initially set to 100. Two processes P1 and P2 run concurrently, and access this shared variable `Balance` without any form of synchronization. Each process performs both a debit and a credit operation, guarded by a conditional check on the current value of `Balance`.

The code executed by P1 is:

```
if (Balance ≥ 70)
{
        Balance = Balance – 70;
        Balance = Balance + 40;
}
```

The code executed by P2 is:

```
if (Balance ≥ 50)
{
        Balance = Balance + 60;
        Balance = Balance – 30;
}
```

Each process runs exactly once, and the update operations on `Balance` are not atomic. Is it possible for the final value of `Balance` (after completion of P1 and P2) to become 130? If yes, show a valid interleaved execution of P1 and P2, that leads to this value. Below, clearly show the interleaving of the execution of the statements of P1 and P2. **[4]**

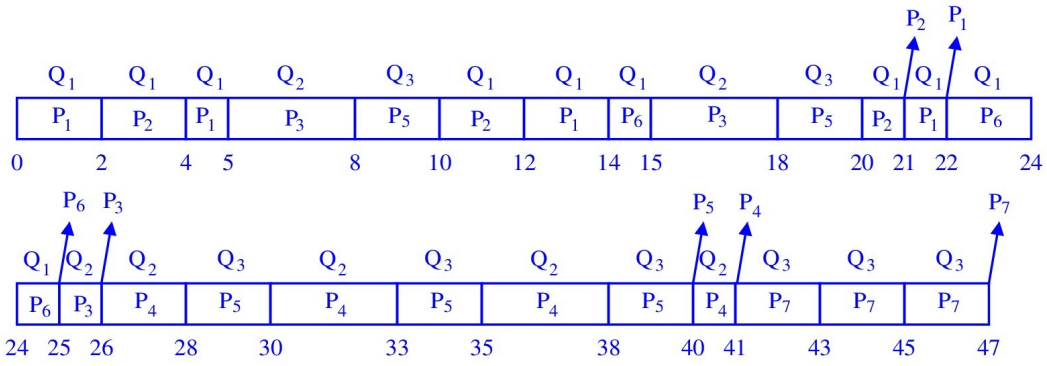
Step	Statements executed by P1	Statements executed by P2	Balance value
1	if (Balance ≥ 70) → true		100
2		if (Balance ≥ 50) → true	100
3		Read Balance = 100 (to execute balance + 60)	100
4	Balance = Balance − 70		30
5	Balance = Balance + 40		70
6		Add and Write Balance = 160	160
7		Balance = Balance − 30	130

2. Consider a Multi-level Queue Scheduling algorithm with three ready queues Q1, Q2, and Q3. Each process is permanently assigned to one queue, and is not allowed to migrate between queues. A distinct scheduling policy is implemented for each ready queue. Precisely, Q1 uses Round Robin scheduling with a time quantum of 2 ms, Q2 uses Preemptive Shortest Remaining Time First (SRTF) scheduling (assume that the CPU burst times of processes in Q2 are known beforehand), and Q3 uses First-Come-First-Served (FCFS) scheduling. To prevent starvation, the OS implements a separate Round Robin mechanism across the three ready queues, referred to as Queue Round Robin (QRR) scheduling. In QRR, each queue (say, Q_x) is assigned a fixed CPU time, say, q_x. All processes in Q_x are allowed to execute for a total duration of at most q_x ms, following the scheduling policy of Q_x. If the allocated CPU time q_x expires while a process is running, the process is preempted and inserted to its corresponding ready queue Q_x. For Q1, insertion is at the back; for Q3, insertion is at the front; and for Q2, insertion is with respect to the remaining burst time of the preempted process. In one round of QRR scheduling, (processes in) queue Q1 first receive 5 ms of CPU time, then (processes in) queue Q2 receive 3 ms of CPU time, and finally (processes in) Q3 receive 2 ms CPU time. If a queue becomes empty before its allocated CPU time expires, the CPU is immediately reassigned to the next queue in the scheduling order.

Consider the following table with seven processes, their arrival times, CPU burst times, and queue allocations. Draw the Gantt chart for the execution of the above processes, clearly indicating the queue (Q1, Q2, or Q3) of the executing process, and termination of each process. Calculate the waiting time for each process, and the average waiting time.

[4+2]

Process	Arrival Time (ms)	CPU Burst Time (ms)	Queue
P1	0	6	Q1
P2	1	5	Q1
P3	2	7	Q2
P4	3	9	Q2
P5	4	10	Q3
P6	6	4	Q1
P7	7	6	Q3



3. Two cooperative processes P_0 and P_1 are running concurrently, and share a buffer capable of storing only one item. P_0 keeps on producing items, but multiple items cannot be stored in the buffer. So the other process P_1 must read each item from the buffer before P_0 writes the next item to the buffer. Writing an item to the buffer by P_0 and reading an item from the buffer by P_1 should be mutually exclusive. In order to guard their critical sections and to alternate their turns, the processes use a variant of Peterson's algorithm, shown below. The algorithm is given for P_i ($i = 0$ or 1). The other process is called P_j , where $j = 1 - i$. Assume that the compiler or the hardware makes no instruction swaps. Determine whether this algorithm satisfies all the requirements of the critical-section problem. Justify your answer. **[5]**

Process P_i :

```
do {
    flag[i] = true;
    turn = j;
    if (flag[j])
        while (turn == j);
    /* critical section */
    flag[i] = false;
    /* remainder section */
} while (true);
```

Doesn't satisfy progress. Assume process *j* is inside the CS. For process *i*, flag[*j*] is checked only once, and then process *i* sits in a tight loop inside while. Once process *j* comes out from CS, still process *i* won't be allowed to enter CS.

4. Consider a uni-processor system running four processes T1, T2, T3, and T4. Each process starts with a CPU burst, followed by an I/O burst, and then followed by a second CPU burst. Each process is assigned a priority (lower number indicates higher priority). Assume that the system implements preemptive priority scheduling combined with round robin scheduling. That is, the CPU scheduler uses priority scheduling, where processes with the same priority run round robin with a time quantum of 8ms. Consider the processes, their CPU and I/O bursts, and priorities as follows.

Process	Arrival Time	First CPU Burst (ms)	I/O burst (ms)	Second CPU burst (ms)	Priority
T1	0	10	30	11	1
T2	6	20	19	17	3
T3	14	20	10	19	3
T4	32	40	20	10	2

Draw the Gantt chart depicting the execution of each process, and show the interval(s) where CPU remains idle. In the Gantt chart, clearly indicate the completion of the first and second CPU burst of each process. Finally, compute the average turnaround time. **[4+1]**