Department of Computer Science BOSTON UNIVERSITY

CS-350 - Fundamentals of Computing Systems

Final Exam
Fall 2022

Name: ____

BU Username: ____________________ BU ID: ____________________

NOTE: *Please try to answer the questions using the provided space as much as possible. A spare blank page is provided at the end of the exam, and an extra page with spare grids is also provided. If you do use this extra page or any spare grid, make sure you reference them properly in your solutions.*

Problem #1:	/24
Problem #2:	/18
Problem #3:	/18
Problem #4:	/18
Problem #5:	/16
Problem #6:	/16
Total Score:	/110

Remarks:

- This is a closed-book/notes exam.
- Basic calculators are allowed.
- You have 120 minutes to complete your exam.
- There are 110 total points.
- If your score is more than 100, it is capped at 100.
- Show your work for full marks.
- Problems and sub-problems weighted as shown
- **Explain all your assumptions clearly.**

Problem 1

Label each of the statements below with either **True (T)** or **False (F)**:

	Statement	**T/F**
a.	Two single-processor systems A and B have identical input traffic and mean service time. If the standard deviation of the service time in A is less than in B, A is expected to exhibit lower response time.	
b.	A discrete event simulator cannot be used to study the behavior of queuing systems with limited queue size, like an M/M/1/K system.	
c.	The input traffic to an M/M/1 queuing system must be Poisson-distributed. In which case, the output traffic will also be Poisson-distributed.	
d.	In a complex system, the resource which constitutes the bottleneck may vary over time if the behavior of the users changes.	
e.	A typical magnetic disk can be modeled as a stateful resource, and the same is true for a traditional DRAM memory controller.	
f.	In global real-time multi-processor scheduling, tasks are never allowed to migrate from one processor to another to prevent deadline misses.	
g.	In a system where real-time scheduling is performed, no more than two processes can be in the *ready* state at the same time.	
h.	Performing a *signal* operation on a semaphore might trigger a state transition from *blocked* to *ready* in some process.	
i.	Like Dekker's Algorithm, the Peterson's Algorithm guarantees 2-party mutual exclusion on in-order CPUs.	
j.	FIFO and SJN are non-preemptive and work-conserving schedulers, therefore they always perform identically in terms of average response time.	
k.	If resource A has higher availability than B, then A will likely operate continuously and correctly over a longer time window compared to B.	
l.	Consider two jobs that have waited inside the ready queue for the same amount of time before being processed. Their slowdown might still be different.	
m.	If the average response time of a resource at steady-state is T_q, then its average throughput can be computed as $1/T_q$.	
n.	Only at most 4 processes can be involved in a deadlock.	
o.	EDF-FF is always capable of producing a valid task-to-CPU assignment for tasksets where RM-FF fails to do so.	
p.	EDF and RM are both instances of static-priority real-time schedulers.	
q.	If a process is unable to enter a critical section protected by a spinlock, it will perform busy-waiting until it is able to do so.	
r.	When using Exponentially Weighted Moving Averages (EWMA) with parameter $\alpha = 0.5$, only the most recent 5 observations determine the next predicted value.	

Note: There are 18 questions. A correct answer will get you 2 points; an incorrect answer -1 points; a blank answer 0 points. The score is capped at 24.

Problem 2

A system accepts requests from a pool of infinite users. These have been found to submit, in total, 2000 requests per second. The inter-arrival time between any two consecutive requests is exponentially distributed.

All the requests initially arrive at the pre-processing (PRE) server. The PRE server is a dual-processor machine that uses a single global queue to hold pending requests. The next request in order of arrival is dispatched for service to one of the two processors as soon as one becomes available. The service time of each request is exponentially distributed with mean 0.6 milliseconds.

Only after completing service at the PRE server the request type is known by the system. With 80% probability, the current request needs *incremental* processing. In this case, it is routed to the single-processor INC machine. Service at the INC server takes, on average, 0.36 milliseconds. After completing service at the INC server, 55 out of 100 requests (on average) require no further service and leave the system. Otherwise, upon leaving the INC server, they are routed back to the PRE machine as if they were new requests.

If, after processing at the PRE stage, the request is determined to be of *bulk* type, it is routed to the BLK server. The BLK server is a single-processor machine which takes on average 1.58 milliseconds to serve a request. After completing processing at the BLK machine, a request always leaves the system.

(a) **[3 points]** Provide a diagram of the system which shows the inter-connections between the PRE, INC, and BLK machines. Describe the queuing models used for each machine, the known parameters, and the assumptions you will employ to solve the system.

(b) **[5 points]** Which resource (i.e., PRE, INC, or BLK) constitutes the bottleneck of the system? Motivate your answer.

(c) **[5 points]** What is the average response time for a generic request arriving at the system?

(d) **[5 points]** What is the total time spent by a generic request waiting for service inside the system, on average?

Problem 3

A traditional magnetic disk is shared by **two** processes, namely P1 and P2. The disk scheduler can be process-aware, in which case it must keep track of which process has submitted each pending request.

Table 1 reports a list of requests submitted by P1 and P2, along with their target cylinder ID and arrival time (in μs).

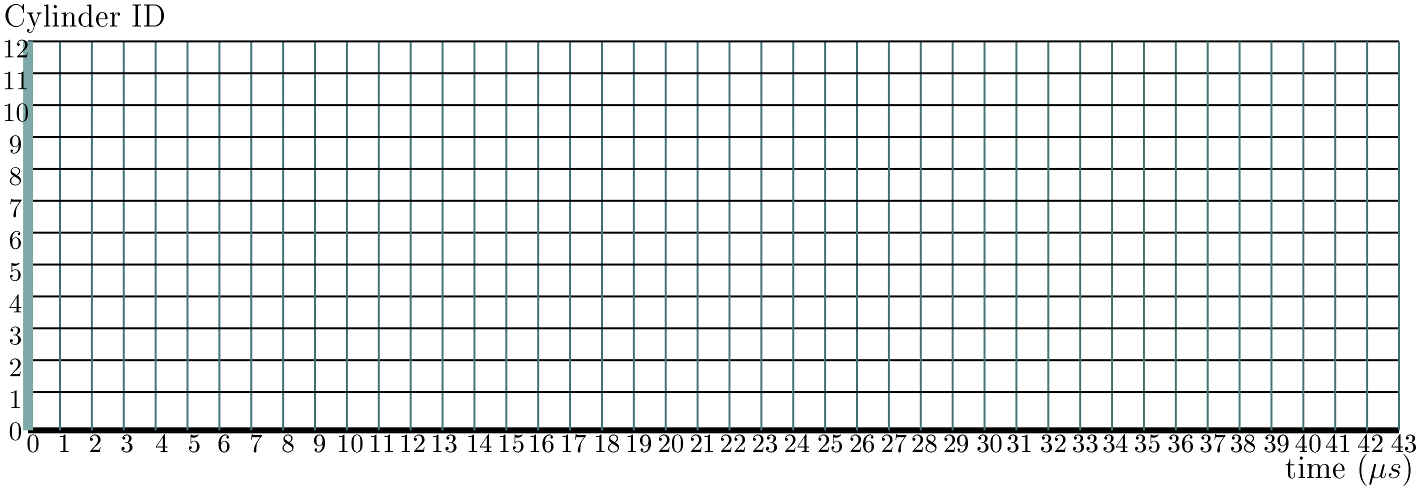
At time 0, the disk head is positioned at cylinder 5. The disk has 13 cylinder in total, numbered 0 through 12. Once the disk head is in position, it takes 0 μs to serve a request. The disk head can move by 1 cylinder for every μs.

If any of the schedulers you will consider below develops any tie, break ties by **lowest process ID first**.

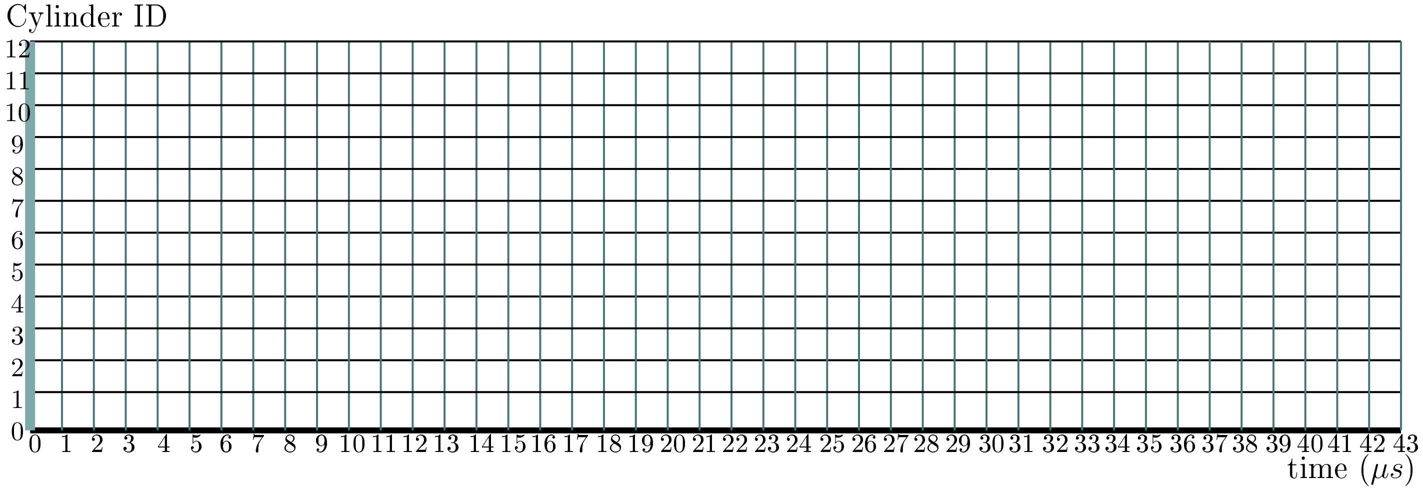
	P1 Requests		**P2 Requests**	
#	**Arrival (us)**	**Cylinder ID**	**Arrival (us)**	**Cylinder ID**
1	1	2	1	6
2	2	3	2	7
3	3	8	6	4
4	7	0	8	6
5	9	0	9	7

Table 1: List of disk requests submitted by P1 and P2.

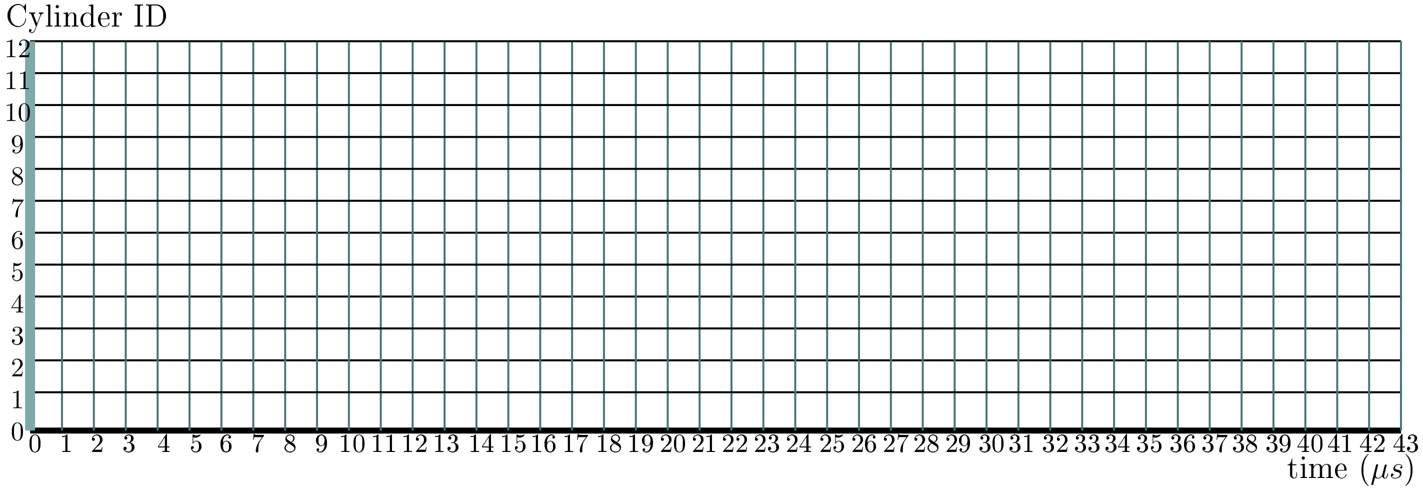
(a) **[6 points]** A pending request is considered "ready" if, at the time when the scheduling decision is made, it targets a cylinder ID which is within ±1 from the current disk head position. Use the grid below to draw the schedule produced by the FR-FCFS scheduler. The FR-FCFS is NOT process-aware, so it will consider all the requests together regardless of the originating process.



(b) [6 points] Schedule the pending disk requests using Priority-based SCAN. If any P1's pending request exists when the scheduling decision is made, P2's requests are ignored and P1's request are served following SCAN. P2's requests are also served following SCAN but only if no P1's requests are pending. Consider an implementation of SCAN in which the disk head does not need to reach the top/bottom cylinder ID in order to switch direction.



(c) [6 points] Schedule the pending disk requests using Round Robin scheduling. The scheduler keeps P1's and P2's requests in two separate queues and in their order of arrival. It then serves a SINGLE request for the process that has been waiting the longest since the last time one of its requests was served.

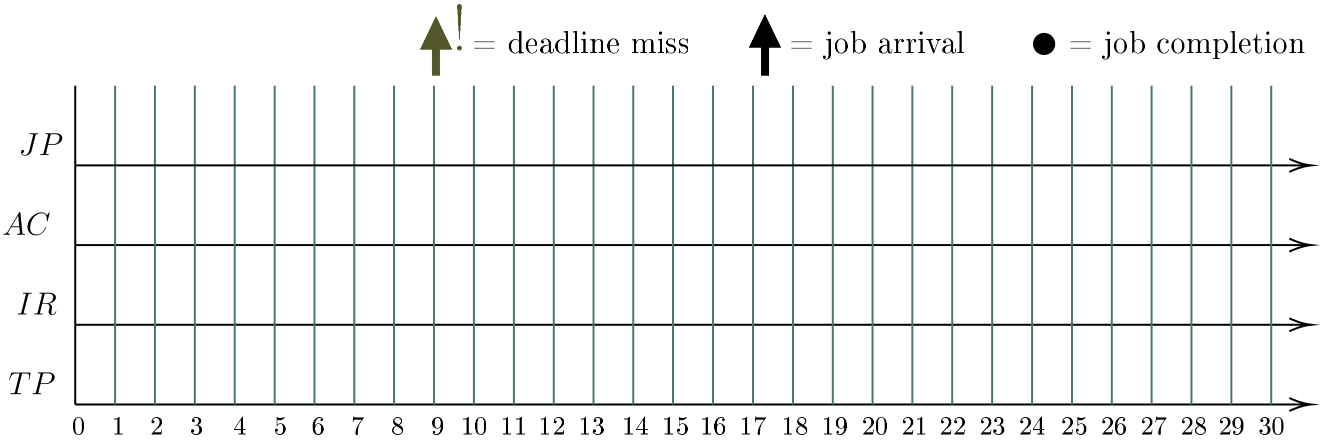


Problem 4

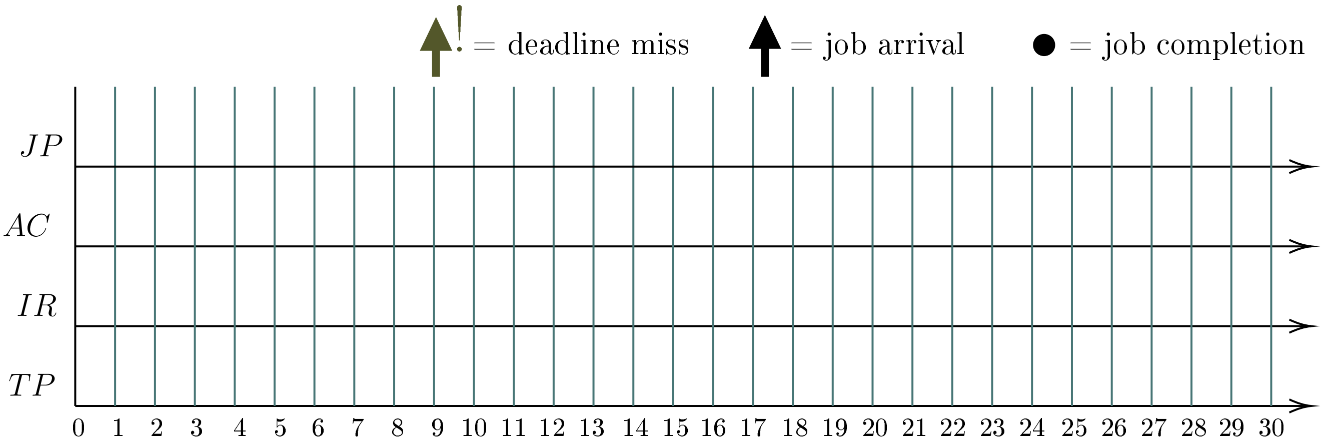
A robotic surgery machine uses four periodic tasks to control the trajectory of the multi-joint robotic arm that performs incisions. Joint position (JP) is sensed at a rate of 200 Hz and it takes 2 millisecond to be performed. Next, actuation commands (AC) to all the joints are forwarded at a rate of 100 Hz, taking a total of 2 milliseconds to complete. Image recognition (IR) to detect safe operation boundaries is carried out every 13 milliseconds and takes 2 ms to complete. Finally, trajectory planning (TP) is performed every 15 milliseconds and takes 3 milliseconds.

NOTE: in all the questions below, if you decide that you *need* to draw the schedule, make conclusions about schedulability based on what you observe between time 0 ms and 30 ms.

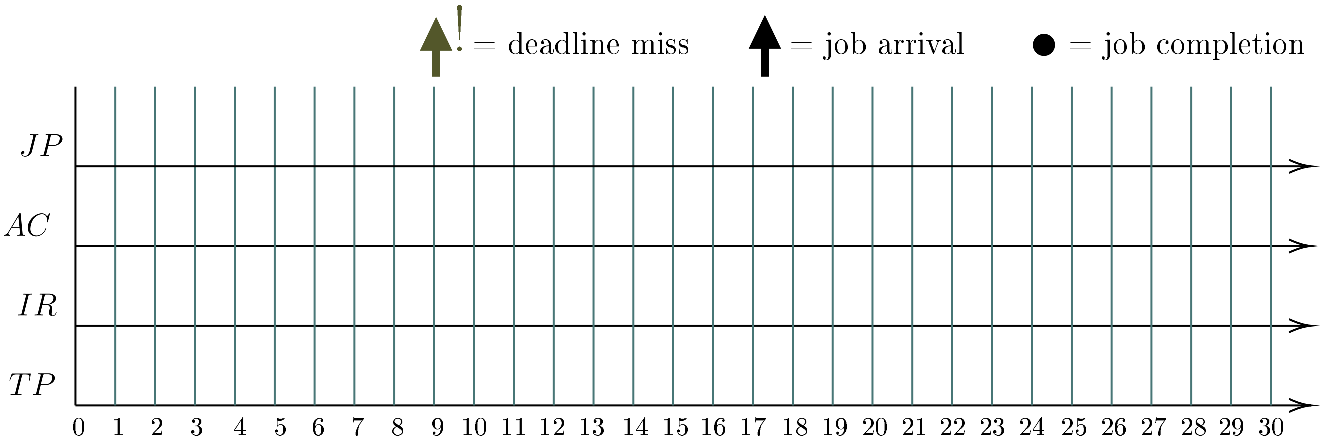
(a) **[4 points]** Assume that we want to schedule the system using Earliest Deadline First (EDF) on a single-processor system. Is the system schedulable? Use the grid below only if strictly needed.



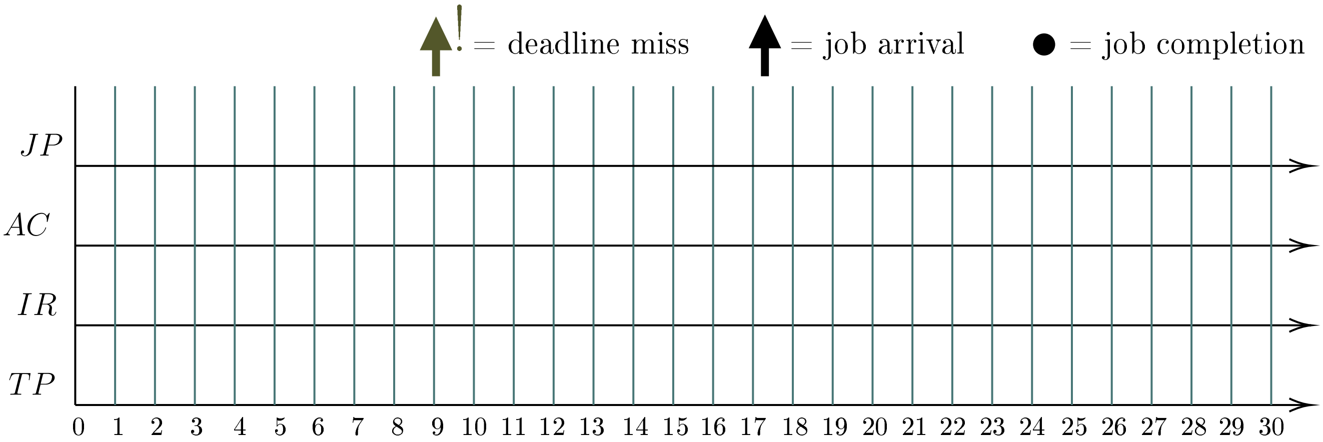
(b) **[4 points]** Assume that we want to schedule the system using fixed-priority preemptive scheduling on a single-processor system. The priority are assigned as follows, from highest priority to lowest priority: JP, TP, AC, IR. Is the system schedulable? Use the grid below only if strictly needed.



(c) **[5 points]** Assume that we want to schedule the system using fixed-priority preemptive scheduling on a single-processor system. The priority are assigned as follows, from highest priority to lowest priority: JP, AC, IR, TP. Is the system schedulable? Use the grid below only if strictly needed.



(d) [5 points] Assume that we want to schedule the system using Non-Preemptive Rate Monotonic (NP-RM) on a single-processor system. This scheduler is identical to RM, except for the fact that preemptions are NOT allowed. Is the system schedulable? Use the grid below only if strictly needed.



Problem 5

What follows are a number of template implementations for multi-producer/single-consumer and multi-producer/multi-consumer interactions between processes. The processes are able to share data structures such as semaphores and queues. But queues need to be appropriately protected against data races if shared between two or more processes.
You are tasked with completing the code to ensure the safe implementation of the intended logic, while maximizing the degree of parallelism. Each blank line can contain at most one statement; *but not all the blank lines might need to be filled.* You also need to provide the initialization value of any variable for which an initialization value is not already provided in the code.

(a) **[8 points]** A first solution involves $N > 1$ Producer processes. All the producers append new integer values in a queue called the `input_queue`. A single Consumer process pops values from the `input_queue`, performs some computation with the retrieved value, and appends the result into a second queue, called `output_queue`. Complete the code of the first solution using semaphores.

```
    Semaphore m = _________________;

    Semaphore val_ready = _________________;
```

Listing 1: Initialization code.

```
Process Producer:
   repeat: /* Repeats forever */

      __________________________;
      int new_val = generate_value();

      __________________________;
      input_queue.append(new_val);

      __________________________;

      __________________________;

   forever


```

Listing 2: Producer code.

```
Process Consumer:
   repeat: /* Repeats forever */

      __________________________;

      __________________________;
      int in_val = input_queue.pop();

      __________________________;
      int result = compute(in_val);
      output_queue.append(result);

      __________________________;

      __________________________;
   forever
```

Listing 3: Consumer code.

(b) **[8 points]** A second solution involves $N > 1$ Producer processes as well as $M > 1$ Consumer processes. Once again, the producers append new integer values in a queue called the `input_queue`. Consumer processes pop values from the `input_queue`, performs some computation with the retrieved value, and appends the result into a second queue, called `output_queue`. Complete the code of the first solution using semaphores.

```
      Semaphore m1 = _________________;

      Semaphore m2 = _________________;

      Semaphore val_ready = _________________;
```

Listing 4: Initialization code.

```
Process Producer:
   repeat: /* Repeats forever */

      _________________________;
      int new_val = generate_value();

      _________________________;
      input_queue.append(new_val);

      _________________________;


      _________________________;

   forever





```

Listing 5: Producer code.

```
Process Consumer:
   repeat: /* Repeats forever */

      _________________________;


      _________________________;

      int in_val = input_queue.pop();

      _________________________;
      int result = compute(in_val);

      _________________________;
      output_queue.append(result);

      _________________________;


      _________________________;
   forever
```

Listing 6: Consumer code.

Problem 6

Consider as system with 5 processes $P_1, \ldots P_5$ sharing 3 resources $R_1, \ldots, R_3$.

(a) [5 points] The immutable system parameters are provided in Table 2, while the state of the system at a given point in time is provided in Table 3. Complete the tables with the missing parameters.

		Parameter	Resources		
			R_1	R_2	R_3
		$R(k)$			
Processes	P_1	$C_1(k)$	0	1	2
	P_2	$C_2(k)$	7	5	0
	P_3	$C_3(k)$	3	5	6
	P_4	$C_4(k)$	6	5	2
	P_5	$C_5(k)$	6	5	6

Table 2: Static parameters for the considered system.

		Parameter	Resources			Parameter	Resources		
			R_1	R_2	R_3		R_1	R_2	R_3
		$V(k)$	5	2	0				
Processes	P_1	$A_1(k)$	0	1	2	$N_1(k)$			
	P_2	$A_2(k)$	0	0	0	$N_2(k)$			
	P_3	$A_3(k)$	3	5	4	$N_3(k)$			
	P_4	$A_4(k)$	6	3	2	$N_4(k)$			
	P_5	$A_5(k)$	0	1	4	$N_5(k)$			

Table 3: System state for considered system.

(b) [5 points] Determine if the current state would be deemed safe by the Banker's Algorithm for deadlock avoidance. Provide your reasoning for full marks.

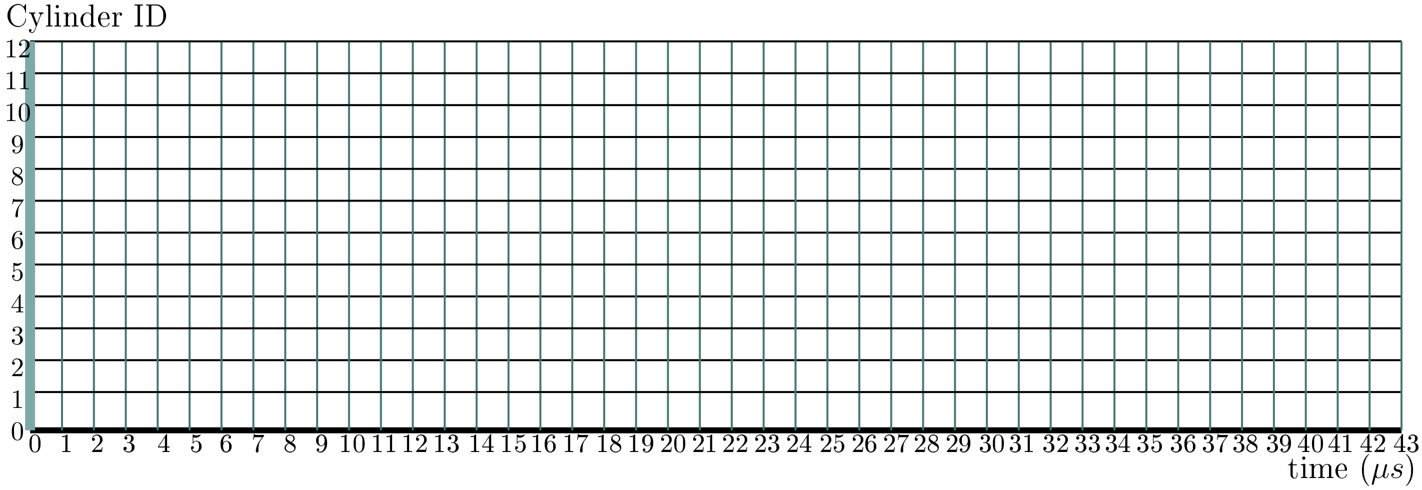
(c) [6 points] While the system is in the state described by Table 2 and 3, P_5 submits an allocation request for 4 units of R_1, 2 unit of R_2, and 0 unit of R_3. Can the request be safely granted? *You may use Table 4 to answer the question.*

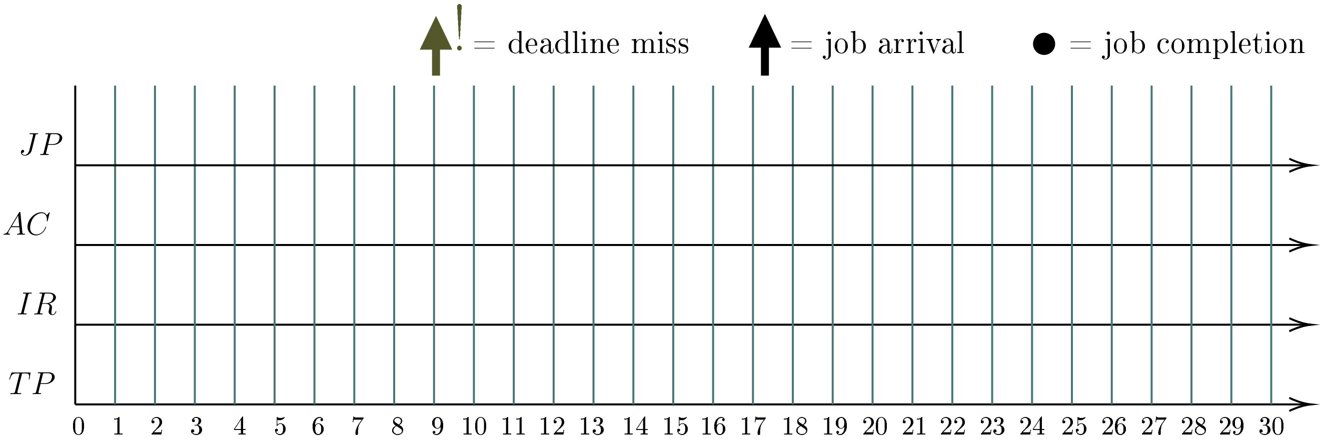
		Parameter	Resources			Parameter	Resources		
			R_1	R_2	R_3		R_1	R_2	R_3
		$V(k)$							
Processes	P_1	$A_1(k)$				$N_1(k)$			
	P_2	$A_2(k)$				$N_2(k)$			
	P_3	$A_3(k)$				$N_3(k)$			
	P_4	$A_4(k)$				$N_4(k)$			
	P_5	$A_5(k)$				$N_5(k)$			

Table 4: System State.

[EXTRA BLANK PAGE]

[EXTRA GRIDS. Reference appropriately if used.]





Some Probability Density Functions

- Poisson: $f(x) = \frac{\lambda^x}{x!}e^{-\lambda}$
- Exponential: $f(x) = \lambda e^{-\lambda x}$, $F(x) = 1 - e^{-\lambda x}$
- Standard Normal: $f(z) = \frac{1}{\sqrt{2\pi}}e^{-0.5z^2}$

Equations for Some Queuing Systems

- M/G/1 System: $q = \frac{\rho^2 A}{1-\rho} + \rho$, $w = \frac{\rho^2 A}{1-\rho}$, where $A = \frac{1}{2}\left[1 + \left(\frac{\sigma_{T_s}}{T_s}\right)^2\right]$
- M/D/1 System: $q = \frac{\rho^2}{2(1-\rho)} + \rho$, $w = \frac{\rho^2}{2(1-\rho)}$
- M/M/1/K system: $q = \begin{cases} \frac{\rho}{1-\rho} - \frac{(K+1)\rho^{K+1}}{1-\rho^{K+1}} & \text{for } \rho \neq 1 \\ \frac{K}{2} & \text{for } \rho = 1 \end{cases}$,

  $P(\text{“rejection”}) = P(S_K) = \begin{cases} \frac{(1-\rho)\rho^K}{1-\rho^{K+1}} & \text{for } \rho \neq 1 \\ \frac{1}{K+1} & \text{for } \rho = 1 \end{cases}$
- M/M/N System: $q = C\frac{\rho}{1-\rho} + N\rho$, $w = C\frac{\rho}{1-\rho}$,
  where $\rho = \frac{\lambda T_s}{N} = \frac{\lambda}{N\mu}$, $C = \frac{1-K}{1-\rho K}$, and $K = \frac{\sum_{i=0}^{N-1}\frac{(N\rho)^i}{i!}}{\sum_{i=0}^{N}\frac{(N\rho)^i}{i!}}$.

Some Schedulability Tests

- Minimum Slowdown for job j_i at time t: $\frac{t-a_i+C_i}{C_i}$
- m tasks schedulable with RM if: $U \leq m(2^{1/m} - 1)$
- Tasks schedulable with RM-FF on N CPUs if: $U \leq N(\sqrt{2} - 1)$
- Tasks schedulable with EDF-FF on N CPUs if: $U \leq \frac{\beta N+1}{\beta+1}$
  where $\beta = \lfloor \frac{1}{\max_k U_k} \rfloor$

Some Useful Numbers

- $2^{1/2} = 1.414213562$
- $2^{1/3} = 1.25992105$
- $2^{1/4} = 1.189207115$
- $2^{1/5} = 1.148698355$
- $2^{1/6} = 1.122462048$
- $2^{1/7} = 1.104089514$
- $2^{1/8} = 1.090507733$
- $2^{1/9} = 1.080059739$