Tentamen 30 Maart 2020

Real-time Systems (Technische Universiteit Delft)

Delft University of Technology
Faculty of Electrical Engineering, Mathematics, and Computer Science (EEMCS)

IN4343 – Real-Time Systems
2019/2020 "Practice exam"

Mitra Nasri

Question:	1	2	3	4	5	6	7	8	Total
Points:	5	10	15	15	10	10	20	5	90
Score:									

- This is a closed book exam
- You may use a **simple** calculator only (i.e. graphical calculators are not permitted)
- Write your answers with a black or blue pen, not with a pencil
- **Always justify your answers**. Without justifying your answer, you may not get a point unless stated otherwise.
- EDF and FP refer to the **preemptive** Earliest-Deadline-First and **preemptive** Fixed-Priority scheduling algorithms unless it is explicitly mentioned that they are non-preemptive.
- Assume that there is no preemption overhead unless it is stated explicitly in the question.
- Assume that a smaller priority indicates a higher priority for the FP scheduling algorithm.

- The exam covers the following material:
  (a) chapters 1-9 of the book "Hard Real-Time Computing Systems (3rd ed)" by G. Buttazzo
  (b) the paper "The Worst-Case Execution-Time Problem" by Wilhelm et al. (except Section 6)
  (c) the paper "Non-Work-Conserving Non-preemptive Scheduling: Motivations, Challenges, and Potential Solutions" by Mitra Nasri and Gerhard Fohler
  (d) the paper "An Exact and Sustainable Analysis of Non-Preemptive Scheduling" by Mitra Nasri and Björn B. Brandenburg

IN4343 Real-Time Systems: Cheat Sheet

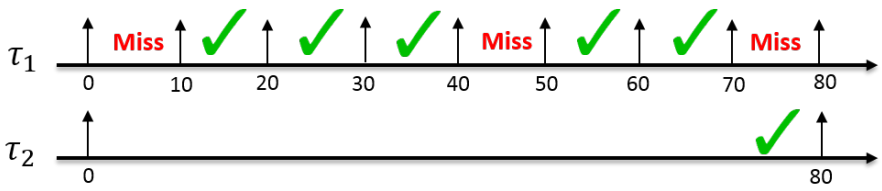
Liu and Layland (LL) test	$U \leq n(2^{1/n} - 1)$
Hyperbolic bound (HB) test	$\prod_{i=1}^{n}(U_i + 1) \leq 2$
Response-Time Analysis (FP) No jitter:	$R_i = C_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_i}{T_k} \right\rceil C_k$
	$R_i^{(0)} = C_i$
With jitter	$WR_i = C_i + \sum_{k=1}^{i-1} \left\lceil \frac{WR_i + AJ_k}{T_k} \right\rceil C_k$
	$BR_i = C_i + \sum_{k=1}^{i-1} \left(\left\lceil \frac{BR_i - AJ_k}{T_k} \right\rceil - 1 \right)^+ C_k$
	$w^+ = \max(w, 0)$
Demand Bound Analysis	$g(t_1, t_2) = \sum_{r_i \geq t_1}^{d_i \leq t_2} C_i$
	$g(0, L) = \sum_{i=1}^{n} \left\lfloor \frac{L + T_i - D_i}{T_i} \right\rfloor C_i$
Schedulability	$\forall L \in \overline{D}, \quad g(0, L) \leq L$
	$\overline{D} = \{d_k \mid d_k \leq \min(H, \max(D_{max}, L^*))\}$
	$H = lcm(T_1, \ldots, T_n)$
	$D_{max} = \max\{D_i \mid \tau_i \in \tau\}$
	$L^* = \frac{\sum_{i=1}^{n}(T_i - D_i) \cdot U_i}{1 - U}$
CBS Server: Arrival of a job J_k	**if** ($\exists$ a pending aperiodic job) **then** enqueue J_k; **else if** $(r_k + (q_s/U_s) > d_s)$ **then** $\{ q_s \leftarrow Q_s; \quad d_s \leftarrow r_k + T_s; \}$
Budget exhaustion	$q_s \leftarrow Q_s$
	$d_s \leftarrow d_s + T_s$
Response-time NP-FP	pre-requisites: $D \leq T$ and preemptive-schedulable
	$WO_i^{(n)} = B_i + \sum_{k=1}^{i-1} \left(\left\lfloor \frac{WO_i^{(n-1)}}{T_k} \right\rfloor + 1 \right) \cdot C_k$
	$B_i = \max\{C_j \mid \forall \tau_j, P_i < P_j\}$
	$R_i = C_i + WO_i$
	$WO_i^{(0)} = B_i + \sum_{k=1}^{i-1} C_k$
Response-time NP-EDF	$\forall \tau_i, 1 < i \leq n, \forall L, T_1 < L < T_i$:
	$L \geq C_i + \sum_{k=1}^{i-1} \left\lfloor \frac{L-1}{T_k} \right\rfloor \cdot C_k$

Question 1 [5 points]

For the following sub-questions, use the following pattern of deadline misses and deadline hits for task τ_1.

Recall: an (m, k) weakly-hard task is feasible if and only if in any window of consecutive k jobs, at least m jobs meet their deadlines.

Assume that the following pattern repeats forever:



(a) [1 point] What is the smallest k for $m = 2$ such that τ_1 is feasible? Why?

Solution:
$k = 4$ because at the end of the hyperperiod there will be two consecutive deadline misses. Hence, in order to have at least 2 successful jobs, we must consider a window of 4 jobs.

(b) [2 points] What is the largest valid m for $k = 7$ such that τ_1 is feasible?

Solution:
$m = 4$ because in the worst-case, in the circular window of length 7, there are at least 3 misses.

(c) [2 points] If the *sixth job* of τ_1 (released at time 50) was also a **miss**, then what would be the smallest k for $m = 2$ such that τ_1 is feasible?

Solution:
$k = 6$ because in any window of 6 jobs, there will be at least two successful jobs. k cannot be 5 or smaller because in the interval [40, 90], there are 4 misses and only one successful execution. Consequently, k must be at least 6.

Question 2 [10 points]

The following task set is scheduled by EDF.

	C_i	T_i	D_i
τ_1	1	5	5
τ_2	8	24	15
τ_3	6	30	20

(a) [3 points] Obtain L^* and then explain at which time instants the *processor demand analysis* must be performed to evaluate whether or not the task set is schedulable by EDF.

Solution:

$$L^* = \frac{\sum_{i=1}^{n}(T_i - D_i) \cdot U_i}{1 - U} = 18.75$$

$$\begin{aligned} D &= \{d_k | d_k \leq \min(H, \max(D_{max}, L^*))\} \\ &= \{d_k | d_k \leq \min(LCM(5, 24, 30), \max(20, 18.75))\} \\ &= \{d_k | d_k \leq 20\} \\ &= \{5, 10, 15, 20\} \end{aligned}$$

(b) [4 points] Use the processor demand analysis to determine whether or not the task set is schedulable. Note: write down all of your calculations/steps.

Solution:

	g_1	g_2	g_3	$g = g_1 + g_2 + g_3$	$g < L$
$L = 5$	1	0	0	1	true
$L = 10$	2	0	0	2	true
$L = 15$	3	8	0	11	true
$L = 20$	4	8	6	18	true

(c) [3 points] Name at least three conditions that must hold for using $U \leq 1$ test as a *necessary and sufficient* schedulability test for EDF.

Solution:

- $\forall \tau_i, D_i = T_i$
- tasks are independent (there is no precedence constraint)
- tasks are preemptive
- there is no scheduling or context switch overhead in the system
- the system has only 1 processor
- tasks do not self-suspend themselves

Question 3 [15 points]

(a) 5 points Name at least two drawbacks of Non-Preemptive Protocol (NPP) to handle accessing shared resources in real-time systems (2 points). Choose one of the drawbacks and then explain why and how that is an issue. To justify your answer, use examples or schematic diagrams (3 points).

Solution:

- A task could be blocked even if it "may" not access a critical section. **Reason:** As soon as a task enters its critical section, it blocks the processor. Hence, if a higher-priority task arrives in the mean time that can potentially call its critical section, it will be blocked. In other words, the protocol blocks the higher-priority task even if it does not enter its critical section (e.g., due to a conditional branch).
- Tasks may be blocked even if they do not use any shared resource. **Reason:** If another task which accesses a shared resource executes non-preemptively, then it will block all other tasks that arrive in the mean time even if those tasks do not access a shared resource at all.
- Long critical sections delay all high-priority tasks. **Reason:** as soon as a task enters its critical section, it will block all other tasks. If the critical section is very long, then any other high-priority task that arrives in the mean time will be blocked and may miss its deadline.

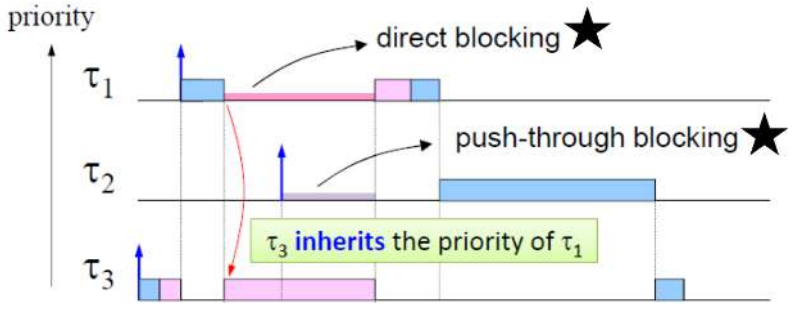
For the examples and diagrams, see slides 24 and 25 of Lecture 9.

(b) 4 points Explain how the Priority Inheritance Protocol (PIP) works.

Solution: It is a resource access protocol in real-time systems in which whenever a task accesses a resource S that is locked by another task, the priority of the **locking task** *upgrades* to the priority of the **highest-priority task that is currently blocked on resource** S.

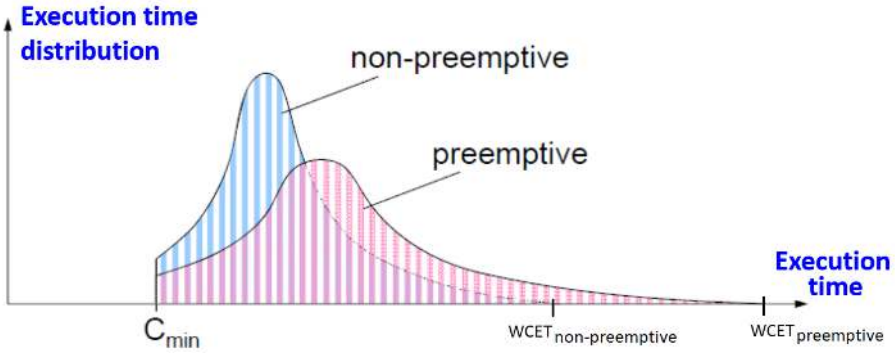
(c) 6 points With a schematic example (figure) show a case where a direct blocking and an indirect blocking happen for a task set when PIP is used. Identify each of these blocking types in your example.

Solution: See slide 34 and 35 of Lecture 9 for more details.



Question 4 [15 points]

Answer the following questions based on the diagram below.



(a) [2 points] Determine whether the following statements are **true** or **false**. (no need for justification)

1. The non-preemptive execution results in a larger *average* and a smaller *maximum* execution time than preemptive execution.
2. A general necessary condition for schedulability of a set of non-preemptive periodic tasks is $\forall i > 1, C_i \leq (T_1 - C_1)$, where τ_1 is the task with the smallest period (i.e., $T_1 = \min\{T_i \mid \tau_i \in \tau\}$).

Solution:

1. **False**. The average execution time of non-preemptive execution is smaller than the average execution time of preemptive case according to the diagram.
2. **False**. The necessary condition is $\forall i > 1, C_i \leq 2 \cdot (T_1 - C_1)$. See the proof in the slides.

(b) [2 points] Why is the minimum execution time (denoted by C^{min}) for both preemptive and non-preemptive execution the same in the above diagram?

Solution: Because in the best case, the task may not be preempted and hence the scenario in which the smallest execution time is observed will be the same for both preemptive and non-preemptive tasks.

(c) [5 points] Name 5 root causes that result in a fatter execution-time distribution curve for preemptive execution (in comparison with the non-preemptive execution).

Solution:

- Context switch overhead
- Cache Related Preemption Delay (CRPD)
- Pipeline costs (context switch will flush the pipeline)
- Bus access cost (e.g., when the bus follows a TDMA policy, after becoming ready again, the preempted task needs to wait until it can access the bus)

- Preemptions may generate more preemptions and hence make the final WCET longer)
- The blocking times caused by accessing critical sections (shared resources)

(d) 3 points According to the diagram, which of the preemptive or non-preemptive execution models results in a lower task utilization? Why?

Solution: Non-preemptive execution will result in a smaller utilization because for the given task:

$$\begin{aligned} & WCET_{NPR} < WCET_{PR} \\ \Rightarrow \quad & \frac{WCET_{NPR}}{T} < \frac{WCET_{PR}}{T} \\ \Rightarrow \quad & U_{NPR} < U_{PR} \end{aligned}$$

(e) 3 points Why even a single periodic task running on a single processor might have different execution times in different runs (jobs)? Name at least 3 reasons.

Solution:

- Change in input value (or a sensor data)
- Taking different program paths, e.g., depending on when the task is executing
- Having input-dependent or event-dependent loop bounds
- Cache memory (may result in different cache hit/miss pattern in different paths)

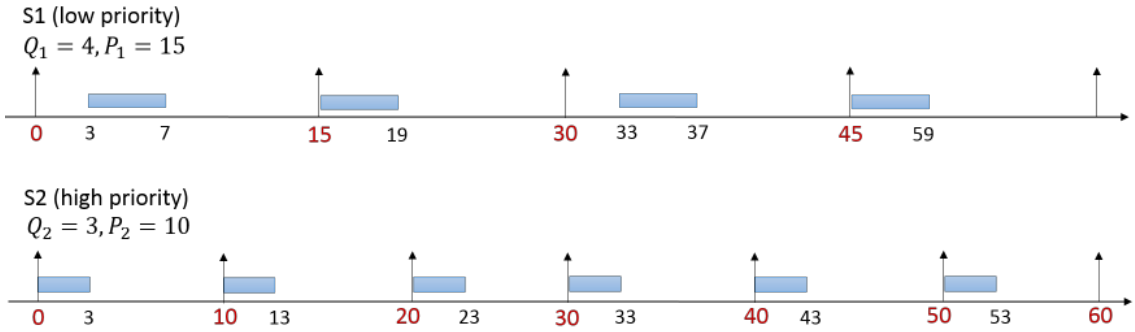
Question 5 [10 points]

Recall that a *supply bound function* $sbf(t)$ is the **minimum amount** of time available in the reservation server S in **every interval** of length t.

For the following questions, assume a system with two periodic servers: $S1$ and $S2$ with budgets $Q_1 = 4$ and $Q_2 = 3$ and periods $P_1 = 15$ and $P_2 = 10$. $S1$ has a lower priority and $S2$ has a higher priority. Both servers activate at time 0 and are scheduled by a fixed-priority scheduling algorithm.

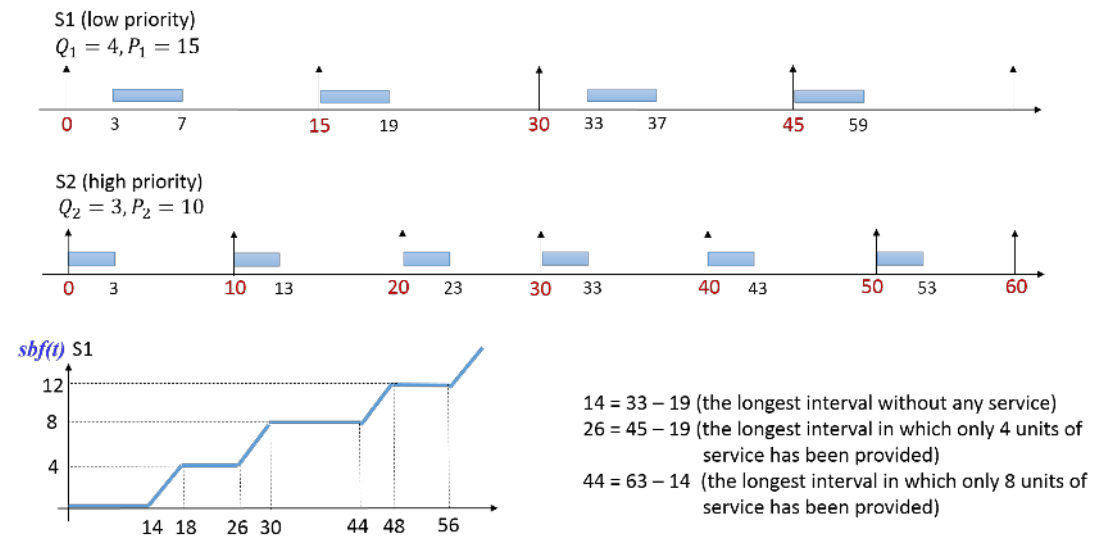
(a) [3 points] Draw the schedule of these two servers for at least two hyperperiods. Note: **pay attention** to the **priorities**.

Solution:



(b) [7 points] **Using the definition** of $sbf(t)$ and the schedule you drew for the first two hyperperiods, draw the supply bound function of $S1$ (as a curve, where the vertical axis shows the $sbf(t)$ and the horizontal axis is t). Notes: Write down the values on the diagram. Do **not** try to approximate the $sbf(t)$ by a line, etc. Just follow the definition.

Solution:

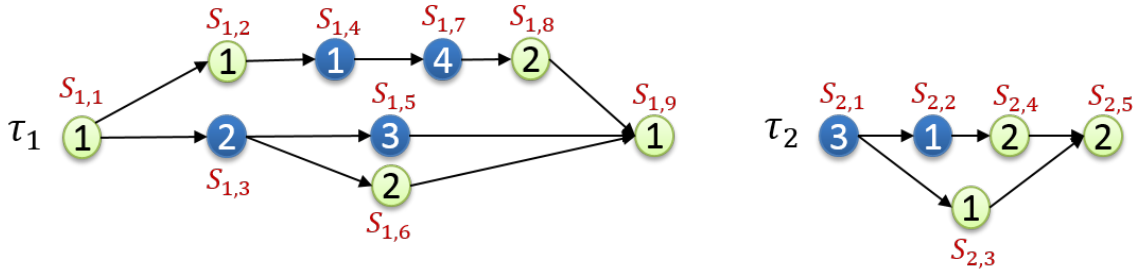


14 = 33 – 19 (the longest interval without any service)
26 = 45 – 19 (the longest interval in which only 4 units of service has been provided)
44 = 63 – 14 (the longest interval in which only 8 units of service has been provided)

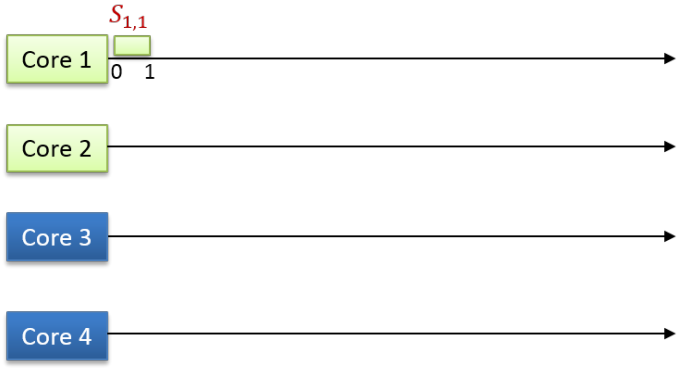
Question 6 [10 points]

Consider a parallel task set shown in the figure below. This task set is scheduled on a multiprocessor platform with 4 cores of two types: 2 type A cores (shown by light green) and 2 type B cores (shown by dark blue). In this system, each subtask can only be executed on a core that has the same type. Subtasks are shown by light green or dark blue circles, where the values inside the circles show the execution time of the subtasks.

The system is scheduled by the global non-preemptive fixed-priority policy, where τ_1 has a higher priority than τ_2. Whenever there is a tie between two subtasks of a task, a subtask with a smaller index will be prioritized.

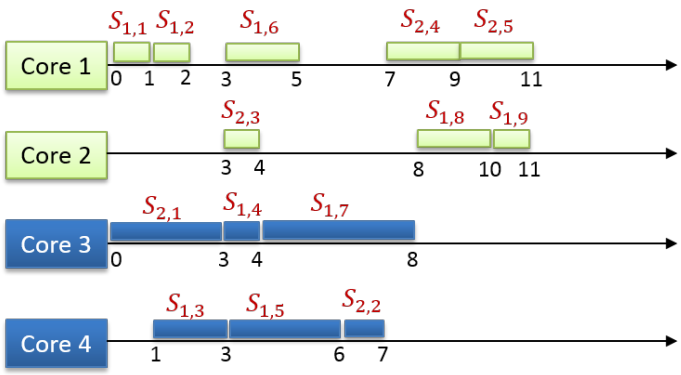


(a) [10 points] Assume that a job of each of these two tasks arrives at time 0. Draw the schedule of these two jobs. Your answer must include a start time, end time, and core number for each of the subtasks of each of the tasks. Any missing information will reduce from your score. You can either draw the schedule or fill a table as shown in the example below.



Subtask	start	end	core
$S_{1,1}$	0	1	Core 1
$S_{1,2}$			
...			

Solution:

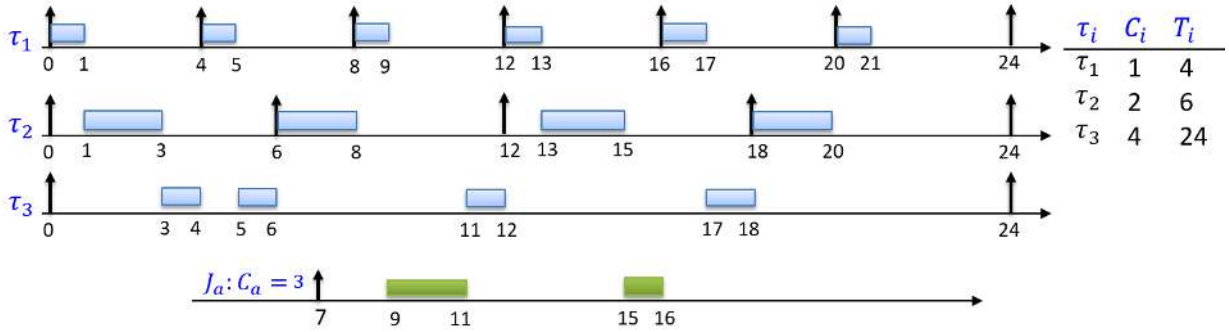


task	id	start	end	core
1	1	0	1	1
	2	1	2	1
	3	1	3	4
	4	3	4	3
	5	3	6	4
	6	3	5	1
	7	4	8	3
	8	8	10	2
	9	10	11	2
2	1	0	3	3
	2	6	7	4
	3	3	4	2
	4	7	9	1
	5	9	11	1

Question 7 [20 points]

The following schedule has been generated by a preemptive *fixed-priority scheduling* algorithm for a system with three periodic tasks and a reservation server. The server and the tasks have been activated at time 0 (there is no offset). All timing parameters (execution times and periods) are *positive integer values* and the total utilization of the system (the three periodic tasks and the server) is *strictly* smaller than 1.

An aperiodic job J_a with 3 units of execution time arrives at time 7 and is executed within the server. The resulting schedule has been shown in the figure below. Based on this schedule, answer the following questions. Note that answers without **proper justifications** may not get any point.



τ_i	C_i	T_i
τ_1	1	4
τ_2	2	6
τ_3	4	24

(a) [1 point] What are the priority relation between the three periodic tasks? **Why?**

Solution: $P_1 < P_2 < P_3$ because of the following reasons:

- At time 0, τ_1 has been scheduled, hence, τ_2 must have a lower priority than τ_1. Namely, $P_1 < P_2$.
- τ_3 could start its execution only at time 3, hence it should have the lowest priority. Consequently, we have $P_1 < P_2 < P_3$.

(b) [4 points] What is the *smallest possible* period and budget of the server in this system? **Why?**

Solution: Period is 5, budget is 1.

Since all timing parameters must be positive integer values, the smallest valid budget will be 1. The total utilization of τ_1, τ_2 and τ_3 is $U' = 0.75$. We have

$$U < 1 \;\Rightarrow\; U' + U_s < 1 \;\Rightarrow\; U_s < 1 - 0.75 \;\Rightarrow\; \frac{Q_s}{T_s} < 0.25 \;\Rightarrow\; \frac{1}{T_s} < 0.25 \;\Rightarrow\; T_s > 4$$

Since the first integer value larger than 4 is 5, the smallest period that the server is 5.

(c) [2 points] What is the priority relation between the server and τ_3? Why?

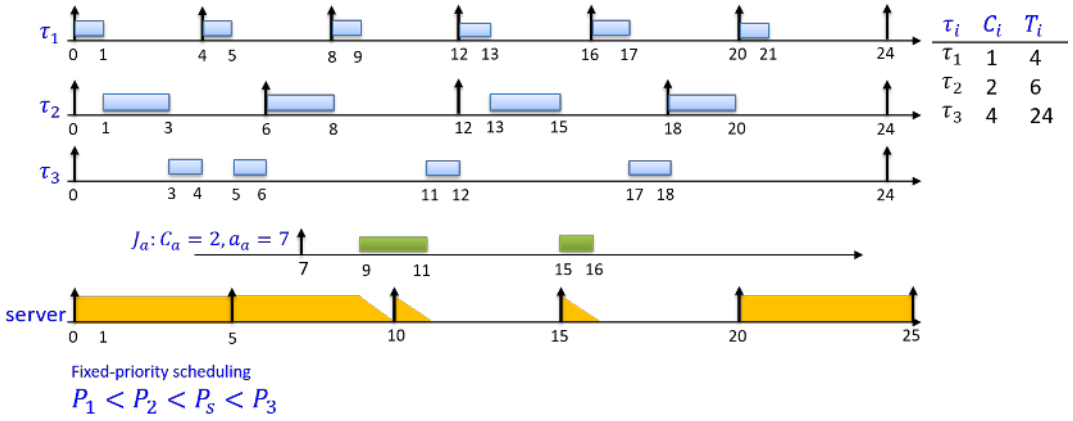
Solution: $P_s < P_3$ because at time 9, the aperiodic job has executed instead of τ_3, so we must have $P_s < P_3$ (since τ_3 has not been completed by time 9).

(d) [4 points] Can the server's period be larger than or equal to 16? Why? Note: assume the following three server types: periodic server, polling server, and deferrable server.

Solution: No. Because if the server does not have at least two activations before time 16, in a fixed-priority schedule, we may not see that τ_3 has been executed in the middle of the two execution intervals of J_a (i.e., the server). The only reason for which τ_3 could have been executed from time 11 to 12 is that the server has run out of budget. Since it regained its budget at time 15, then it must have arrived sometime earlier than or at 15.

(e) [6 points] What are the **type** (periodic, polling, or deferrable), **period**, and **budget** of the server? *Justify your answer by explaining how the resulting schedule could have been created by your proposed server parameters.*

Solution: This question may have multiple correct answers. Here is one of the correct answers: The server is a **deferrable server** with budget 1 and period 5 with the following priorities $P_1 < P_2 < P_s < P_3$. See the figure below for justification.



- During the interval $[7, 9]$, J_a was in the server's queue but the server could not have executed it because the server has a lower priority than τ_2.
- At time 9, the server becomes the highest priority and uses its left-over budget that has been there since time 5 to run one unit of J_a's execution time. Then it runs out of budget.
- At time 10, the server is replenished by the new budget and since there is no higher-priority task in the system, it continues executing J_a. However, its budget finishes at time 11.
- At time 15, the server's budget is replenished again and it can complete the execution of J_a since there is no other higher-priority tasks in the system from time 15 to 16.

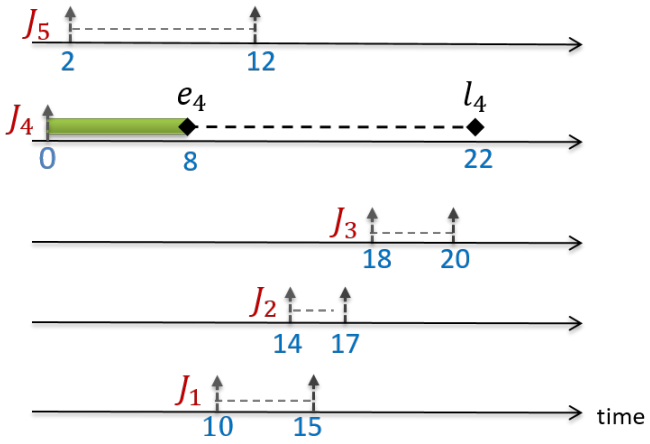
(f) [3 points] Assume that a second aperiodic job J_b arrives at the system at time 16 and has 1 unit of execution time. In which interval would the job J_b execute? Why?

Solution: In the interval $[21, 22]$ because the server budget for the interval $[15, 20]$ has finished after executing the last unit of the execution time of J_a at time 16. As a result, J_b must wait until the next activation of the server.

Question 8 [5 points]

Consider the following situation where in a uniprocessor system that uses non-preemptive fixed-priority scheduling, job J_4 that is released at time 0 has been scheduled on the processor. The table on the left side of the figure provides you the information about the available jobs in this system (including the priority of each job, its best- and worst-case execution times, and its earliest and latest release time).

Job	priority	Execution time Min	Execution time Max	Release time Min	Release time Max
J_1	1 (high)	1	2	10	15
J_2	2	1	2	14	17
J_3	3	1	2	18	20
J_4	4	8	22	0	0
J_5	5 (low)	1	2	2	12



(a) [2 points] Which of the job(s) in this system can never be executed directly after J_4? Justify your answer.

Solution: The answer is J_3 because at the time J_3 can possibly be released, there are two other higher-priority jobs (J_1 nd J_2) that have certainly be released. Thus, J_3 can never be the highest-priority pending job directly after J_4.

(b) [2 points] If J_4 finishes at time 16, which of the job(s) in this system may have a chance to be executed after J_4 before any other job? Justify your answer.

Solution: The answer is J_1 because at the time 16, J_1 has been certainly released and it has the highest priority. Hence, as long as J_1 has not been executed, the underlying fixed-priority scheduler will not schedule any other job.

(c) [1 point] Under what situation J_5 can be scheduled after J_4? Justify your answer.

Solution: One possible answer is: $C_4 = 8, r_5 = 8$. Then, regardless of the release time or execution time of the other jobs, J_5 will be scheduled at time 8 and hence directly follows J_4 because then at time 8, it would be the only job that is in the ready queue and hence will be selected by the underlying work-conserving fixed-priority scheduler.