Overview

- Today: this lecture explains how to analyze schedulability with pre-period deadlines and blocking times. We also introduce aperiodic task scheduling.

- **To learn more on real-time scheduling**:
- **see chapter 4 on “Hard Real-Time Computing Systems” book from G. Buttazzo** (useful chapters are in the Lab!)

The Concept of Blocking in RM scheduling

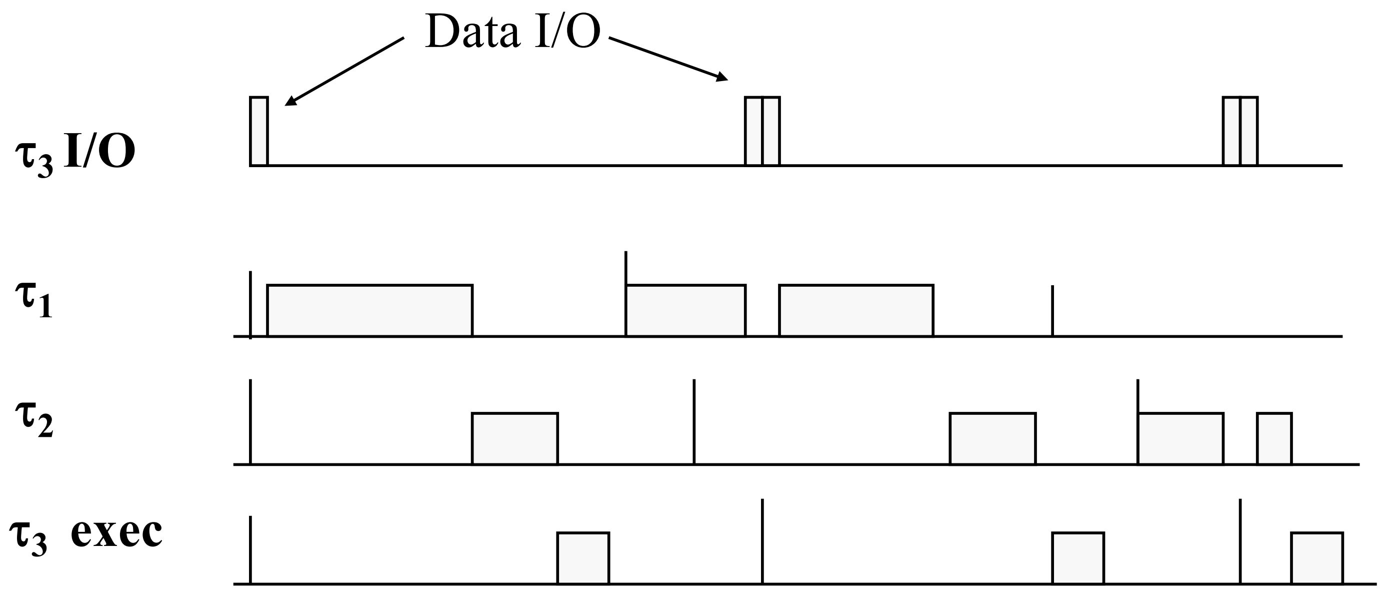
- In rate monotonic scheduling, short period tasks are given higher priorities. When a long period task is delayed by the execution of short period tasks, the long period task is said to be PREEMPTED by short period tasks.

- What if, for some reason, the long period (low-priority) task delays the execution of short period (high-priority) tasks? In this case, the short period (high-priority) task is said to be BLOCKED by long period (low-priority) tasks.

- (NOTE: in OS literature, if a higher priority task delays a lower priority one, it is called preemption, independent of periods.)

Blocking Due to I/O and Interrupt Handling

- In this example, τ_3 has the longest period. However its data I/O executes at top priority to reduce jitter. As a result, τ_3' s data I/O blocks the execution of shorter period tasks τ_1 and $\tau_{2.}$



- Similarly, if we use interrupts to perform the data I/O, the ISR for data I/O will have higher priority even if it is done for a longer period task

Blocking and Pre-periodic Deadlines under RM

- In BOTH Utilization Bound and Exact Analysis, blocking time B can be modeled as if the task has a longer execution time (C+B).

- Assume to have blocking times and pre-period deadlines:
  1. **When using utilization test**, we need to adjust the task utilization: so we inflate c_i to $(c_i + B_i + \Delta_i)$ and check if the task is still schedulable. Suppose that task 2 has B_2 as blocking time, we just add B_2 to task T2's execution time ***LOCALLY***
  2. **When using exact analysis**, we inflate c_i to $(c_i + B_i)$ and check if the finishing time of first job is less than or equal to D_i by using this formula:

$$r_i^{k+1} = c_i + \underbrace{B_i}_{} + \sum_{j=1}^{i-1} \left\lceil \frac{r_i^k}{p_j} \right\rceil c_j, \quad \text{where } r_i^0 = \underbrace{B_i}_{} + \sum_{j=1}^{i} c_j$$

This term models the blocking time

Task switching, pre-period deadline and Blocking

- Suppose that a task has relative deadline D (deadline less than period) and blocking time B, we just add (B+(P-D)) to its execution time when using UB test (remember to inflate execution time <u>LOCALLY</u>). In exact schedulability test, we will inflate the execution time by B and move the deadline from P to D.

$$\tau_1 \qquad \frac{(C_1 + 2S + P_1 - D_1 + B_1)}{P_1} \le U(1)$$

$$\tau_2 \qquad \frac{(C_1 + 2S)}{P_1} + \frac{(C_2 + 2S + P_2 - D_2 + B_2)}{P_2} \le U(2)$$

$$\tau_3 \qquad \frac{(C_1 + 2S)}{P_1} + \frac{(C_2 + 2S)}{P_2} + \frac{(C_3 + 2S + P_3 - D_3)}{P_3} \le U(3)$$

Note that B_3 is always zero. τ_3 is the task with the longest period and therefore it cannot be blocked by a task with longer period.

Class exercise on schedulability analysis

	C	T	B	D
Task $\tau1$	1	4	?	
Task $\tau2$	1	6	?	
Task $\tau3$	4	13	?	12

Suppose that S=0. However, task τ_3 has two parts. part 1 has execution time C_3=1; part 2 is 3 units long and does I/O operations. To reduce jitter, part 2 is executed with the highest priority in the system.

Fill in the blocking times in the table and determine if all three tasks are schedulable?

Class exercise on schedulability analysis

	C	T	B	D
Task τ1	1	4	?	
Task τ2	1	6	?	
Task τ3	4	13	?	12

Class exercise on schedulability analysis

	C	T	B	D
Task $\tau 1$	1	4	?	
Task $\tau 2$	1	6	?	
Task $\tau 3$	4	13	?	12

Class exercise on schedulability analysis

	C	P	B	D
Task τ1	1	4	3	
Task τ2	1	6	3	
Task τ3	4	13	0	12

Utilization Bound with blocking and/or pre-preriod deadlines becomes a task by task test: if you have N tasks, you need to check N equations to verify the schedulability of all the task set (The original formulation studied in lecture 9 allowed to check the schedulability of all task set with a single test!!!).

τ_1 $$\frac{(1+3)}{4} = 1.00 = U(1) = 1.00$$

τ_2 $$\frac{1}{4} + \frac{(1+3)}{6} = 0.916 > U(2)$$

Check with exact analysis!

τ_2 $\mathbf{a_0}$ = 1 + 1 + 3 = 5

$\mathbf{a_1}$ = 1 + 3 + ceil(5/4)*1 = 6

$\mathbf{a_2}$ = 1 + 3 + ceil(6/4)*1 = 6 ➔ schedulable

Class exercise on schedulability analysis

	C	P	B	D
Task τ1	1	4	3	
Task τ2	1	6	3	
Task τ3	4	13	0	12

$\tau 3$ $\mathbf{a_0 =}$ 1+1+4 = 6,

$\mathbf{a_1}$ = ceil(6/4)*1 + ceil(6/6)*1 + 4 = 7

$\mathbf{a_2}$ = ceil(7/4)*1 + ceil(7/6)*1 + 4 = 8

$\mathbf{a_3}$ = ceil(8/4)*1 + ceil(8/6)*1 + 4 = 8 < 12

The lowest priority task is schedulable!!

Class exercise on schedulability analysis

	C	P	B	D
Task τ1	26	59	0	59
Task τ2	10	60	4	50
Task τ3	25	155	5	135
Task τ4	15	210	0	180

2S = 1

Class exercise on schedulability analysis

	C	P	B	D
Task $\tau1$	26	59	0	59
Task $\tau2$	10	60	4	50
Task $\tau3$	25	155	5	135
Task $\tau4$	15	210	0	180

2S = 1

Class exercise on schedulability analysis

	C	P	B	D
Task τ1	26	59	0	59
Task τ2	10	60	4	50
Task τ3	25	155	5	135
Task τ4	15	210	0	180

2S = 1

τ_1 $$\frac{27}{59} < U(1) = 1.00$$

τ_2 $$\frac{27}{59} + \frac{(11+10+4)}{60} = 0.8743 > U(2)$$ ➔ It failed, check with exact test

τ_2 $\mathbf{a_0}$ = 11+ 4 + 27 = 42

$\mathbf{a_1}$ = 11 + 4 + ceil(42/59)*27 = 42 ➔ schedulable

Class exercise on schedulability analysis

	C	P	B	D
Task $\tau 1$	26	59	0	59
Task $\tau 2$	10	60	4	50
Task $\tau 3$	25	155	5	135
Task $\tau 4$	15	210	0	180

2S = 1

τ_3 $\mathbf{a_0}$ = 26+ 5 + 27 + 11 = 69

$\mathbf{a_1}$ = 26 + 5 + ceil(69/59)*27 + ceil(69/60)*11 = 107

$\mathbf{a_2}$ = 26 + 5 + ceil(107/59)*27 + ceil(107/60)*11 = 107 ➔ schedulable

τ_4

$$\frac{27}{59}+\frac{11}{60}+\frac{26}{155}+\frac{16+30}{210}=1.028>100\%$$

Don't stop! Check with exact analysis even if total load is over 100%. This is because pre-period deadline is not real execution time.

Class exercise on schedulability analysis

	C	P	B	D
Task $\tau 1$	26	59	0	59
Task $\tau 2$	10	60	4	50
Task $\tau 3$	25	155	5	135
Task $\tau 4$	15	210	0	180

2S = 1

τ_4

$\mathbf{a_0}$ = 16+ 27 + 11 + 26 = 80

$\mathbf{a_1}$ = 16 + ceil(80/59)*27 + ceil(80/60)*11 + ceil(80/155)*26 = 118

$\mathbf{a_2}$ = 16 + ceil(118/59)*27 + ceil(118/60)*11 + ceil(118/155)*26 = 118

➔ schedulable

Aperiodic tasks: concepts and definitions

Aperiodic task: runs at irregular intervals.

Aperiodic task's deadline can be

- hard, with the following pre-conditions:
  - a lower bound exists on minimum inter-arrival time
  - an upper bound exists on worst-case computing time for the aperiodic

- soft ➔ it does not need pre-conditions.
  - no special requirement on inter-arrival time, typical assumption: exponential inter-arrival time (Poisson Process)
  - no special requirement on worst case execution, typical assumption: exponential execution time

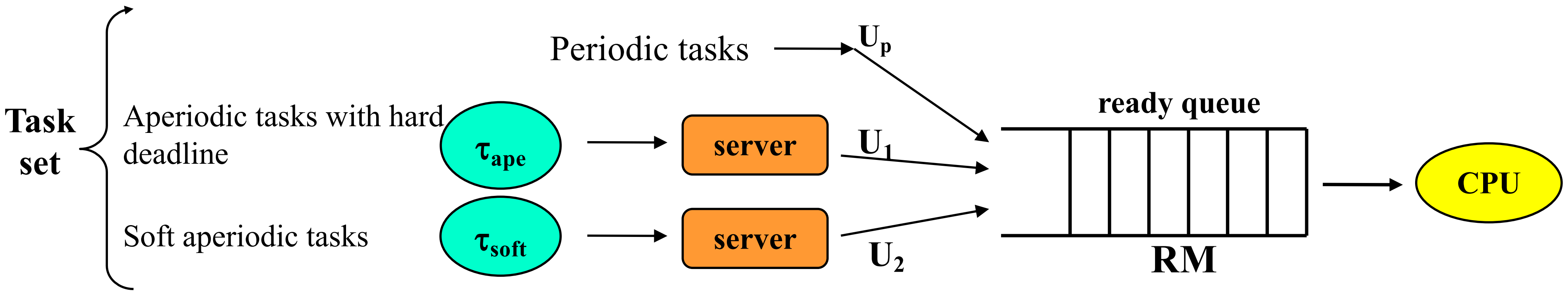
The Fundamental Idea for handling aperiodic tasks

- Rate monotonic scheduling is a periodic framework. To handle aperiodics, we must convert the aperiodic event service into a periodic framework.

- Except in the case of using interrupt handler to serve aperiodics, the basic idea is to periodically allocate CPU cycles to each stream of aperiodic requests. This CPU allocation is called "aperiodic server":
  - Polling server
  - Sporadic server

Types of Aperiodic Requests

- The jobs of an aperiodic task have random release times
  - Soft aperiodic tasks:
    - random arrivals such as a Poisson distribution:
    - the execution time can also be random such as exponential distribution
    - typically it models users' requests.
  - Aperiodic tasks with hard deadline:
    - there is a minimal separation between two consecutive arrivals
    - there is a worst-case execution time bound
    - models emergency requests such as the warning of engine overheat

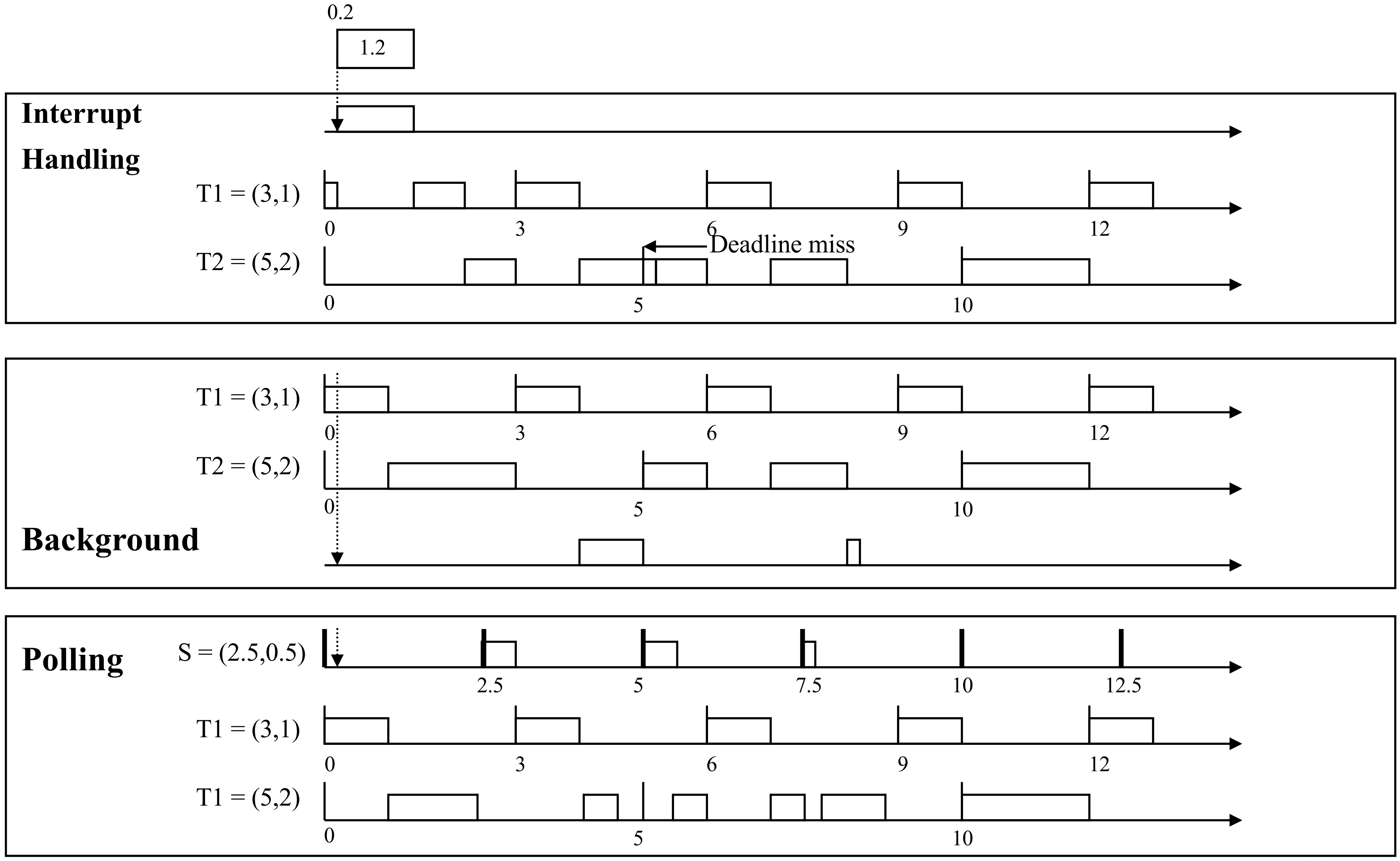


Interrupt Handling or Background Service

- One way to serve aperiodic requests is handle them right at the interrupt handler.
  - This gives the best response time but can greatly impact the hard real-time periodic tasks causing deadline misses.
  - Use it as last resort only say pending power failure exception handling

- Another simple method is to give background class priority to aperiodic requests. This works as well but the response time is not too good. For example:
  - Assign Priority levels 256 to 50 for periodic tasks
  - Assign Priority levels 1 to 49 for aperiodic tasks

Interrupt Handling, Background, Polling



Polling Server - 1

- The simplest form of integrated aperiodic and periodic service is polling server.
  - For each aperiodic task, we assign a periodic service with budget e_s and period p_s. This creates a server (e_s, p_s)
  - The aperiodic requests are buffered into a queue
  - When polling server starts,
    - Resumes the existing job if it was suspended in last cycle.
    - it checks the queue.
  - The polling server runs until
    - All the requests are served
    - Or suspends itself when the budget is exhausted.

  - Remark: a small improvement is to run the tasks in background priority instead of suspend. This background mode can be applied to any aperiodic server.
  - If an aperiodic task arrives after the beginning of the server period, the task has to wait for the beginning of next period before being served.

Polling - 2

- A polling server behaves just like a periodic task and thus the schedulability of periodic tasks is easy to analyze. For example, if we use L&L bound,

$$\sum_{i=1}^{n} \frac{e_i}{p_i} + \frac{e_s}{p_s} \le (n+1)\left(2^{1/(n+1)} - 1\right)$$

Polling - 3

- Main attributes of a Polling Server:
  - it buffers all aperiodic requests in a FIFO queue
  - serve the buffered requests periodically with
    - a budget C
    - and a period P
    - the priority is assigned according to the server period (higher rate, higher priority just like periodic tasks)

  - The utilization of a polling server is simply U=C/P

- NOTE: each time, the server will keep serving buffered requests until either
  - all the buffered requests are serviced (<u>unused budget, if any, will be discarded</u>),
  - or the budget C runs out. In this case, the server suspends until the beginning of next period with a new C budget again.

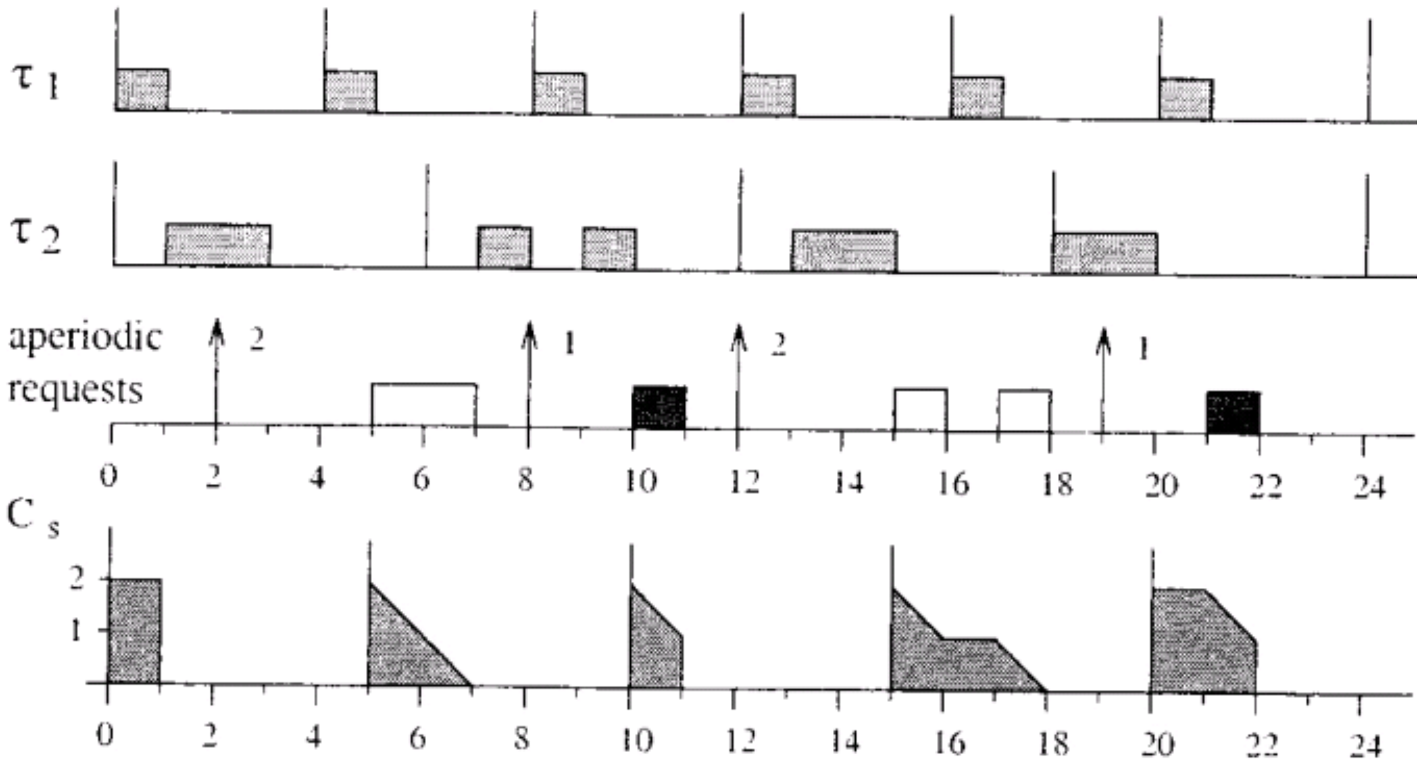
Example with a Polling Server

	C_i	T_i
τ_1	1	4
τ_2	2	6

Server

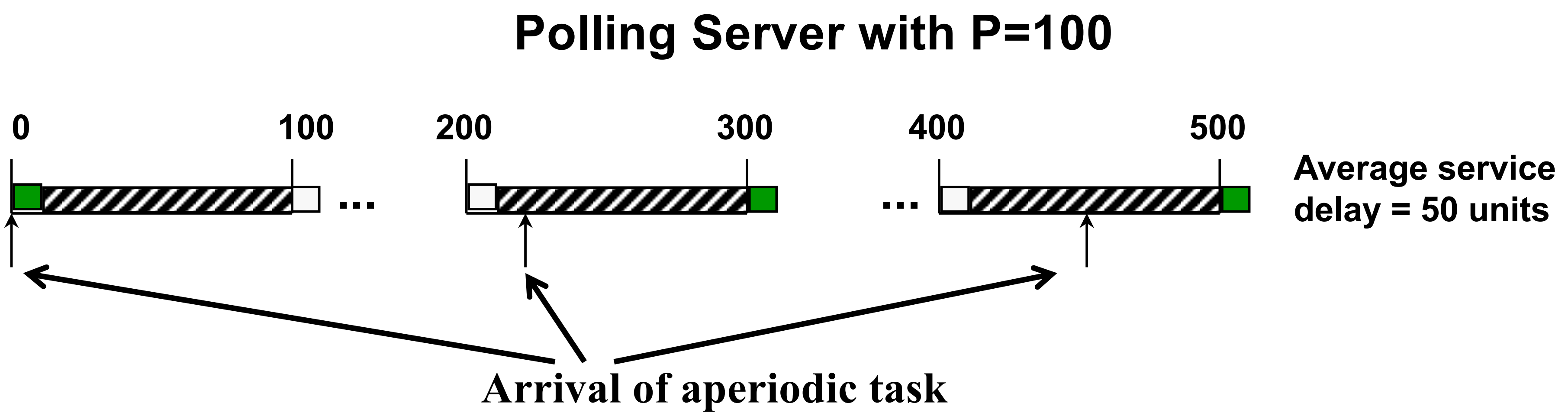
$C_s = 2$

$T_s = 5$



Example of a Polling Server scheduled by RM.

Performance of a Polling Server

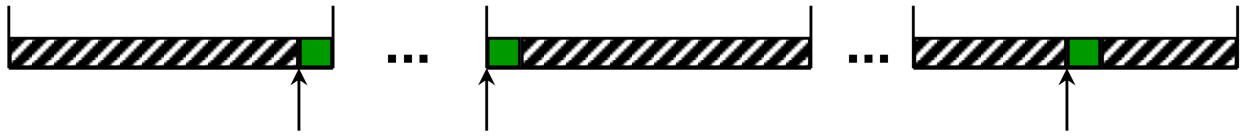

Polling Server with P=100
0
100
200
300
400
500
…
…
Average service delay = 50 units
Arrival of aperiodic task

Service delay of a polling server is, on average, roughly half of the server period.

- **higher polling rate (shorter server period) will give better response time.**
- **low polling rate will have lower scheduling overhead.**

Using Interrupt Handler

Interrupt Handler



Service delay: negligible

- Handle aperiodic requests within interrupt handler gives the best performance, since interrupt handlers run at priority higher than applications
- Precisely for the same reason, a larger amount of such interrupts would cause deadlines of periodic tasks to be missed.
- It is a solution with serious side effects. Use it ONLY as a last resort for short fuse hard deadline aperiodic requests such as power failure warning.

Sporadic Server - 1

- The Sporadic Server (SS) differs from Polling Server in the way it replenishes its capacity. Whereas Polling periodically replenishes its capacity at the beginning of each server period, SS replenishes its capacity only after it has been consumed by aperiodic task execution.

- We will see that Sporadic Server can be treated as if it is a periodic task too. However, SS has better response time than Polling server.

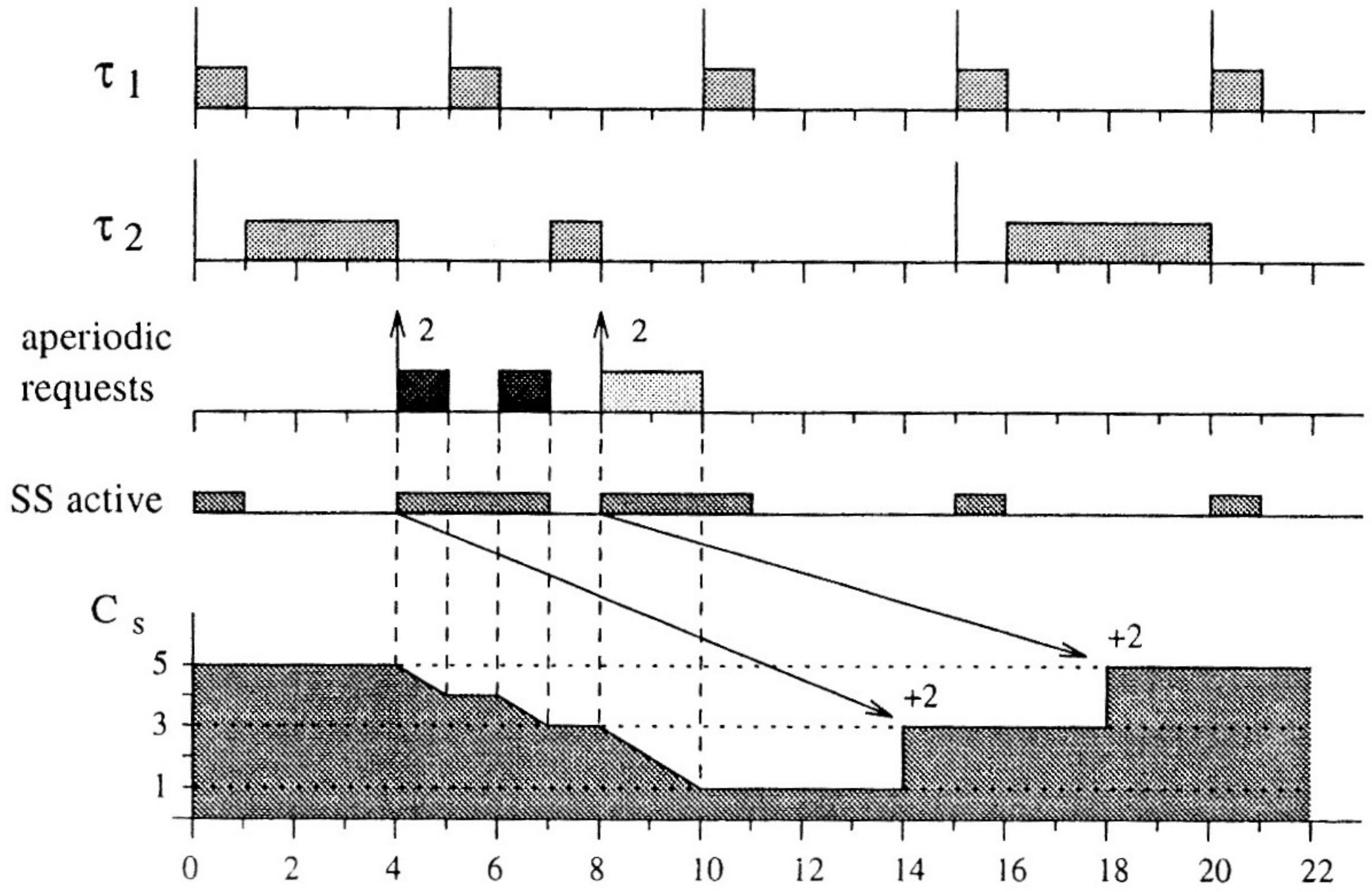
- What is the main advantage of SS?
- If Sporadic Server has the highest priority in the system, it can provide a service delay almost equivalent to an interrupt handler but without causing the deadline miss of other tasks!!!

Sporadic Server - 2

- A Sporadic Server with priority $Prio_s$ is said to be *active* when it is executing or another task with priority $Prio_T \geq Prio_s$ is executing. Hence, the server remains active even when it is preempted by a higher priority task.
- If the server is not active, it is said to be *idle*
- **Replenishment Time (RT):** it is set as soon as SS becomes active and the server capacity $C_s>0$. Let T_A be such a time. The value of RT is set equal to T_A plus the server period ($RT= T_A+ p_s$).
- **Replenishment Amount (RA):** The RA to be done at time RT is computed when SS becomes idle or the server capacity C_s has been exhausted. Let T_I be such a time. The value of RA is set equal to the capacity consumed within the interval $[T_A, T_I]$.

Sporadic Server - 3

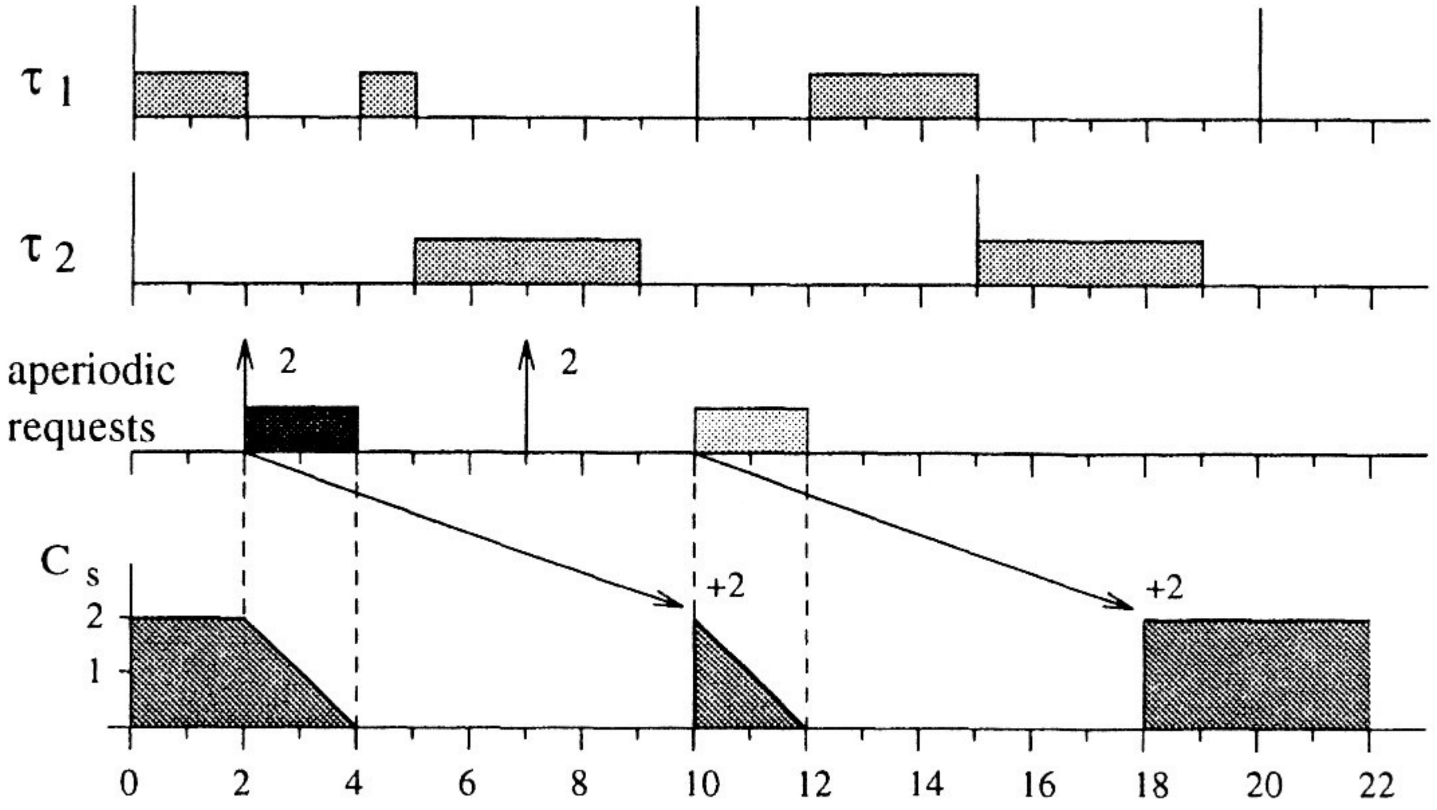
- Example of a medium-priority Sporadic Server.



	C	p
T_1	1	5
T_S	5	10
T_2	4	15

Sporadic Server - 4

- Example of a high-priority Sporadic Server.



	C	p
T_S	2	8
T_1	3	10
T_2	4	15

Sporadic Server - 5

- The Sporadic Server can defer its execution and preserve its budget even if no aperiodic requests are pending. This allows SS to achieve better response time compared to Polling Server.

- *What about the schedulability analysis in the presence of Sporadic Server?*

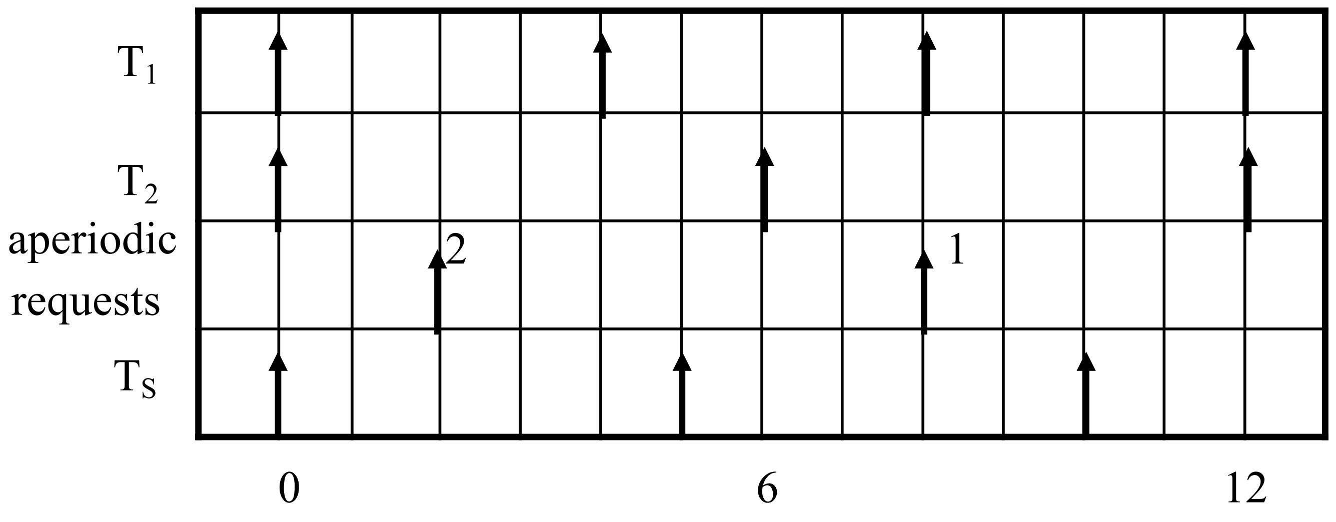
- *A periodic task set that is schedulable with a task T_i is also schedulable if T_i is replaced by a Sporadic Server with the same period and execution time.*

  ➔ In other words, Sporadic Server behaves like a regular periodic task, so nothing changes when you check the schedulability of the task set.

Class exercise – Polling vs Sporadic server

- Consider the following task set
  - T_1 {C_1=1, p_1= 4}
  - T_2 {C_2=2, p_2= 6}
  - T_s {C_s=1, p_s= 5}

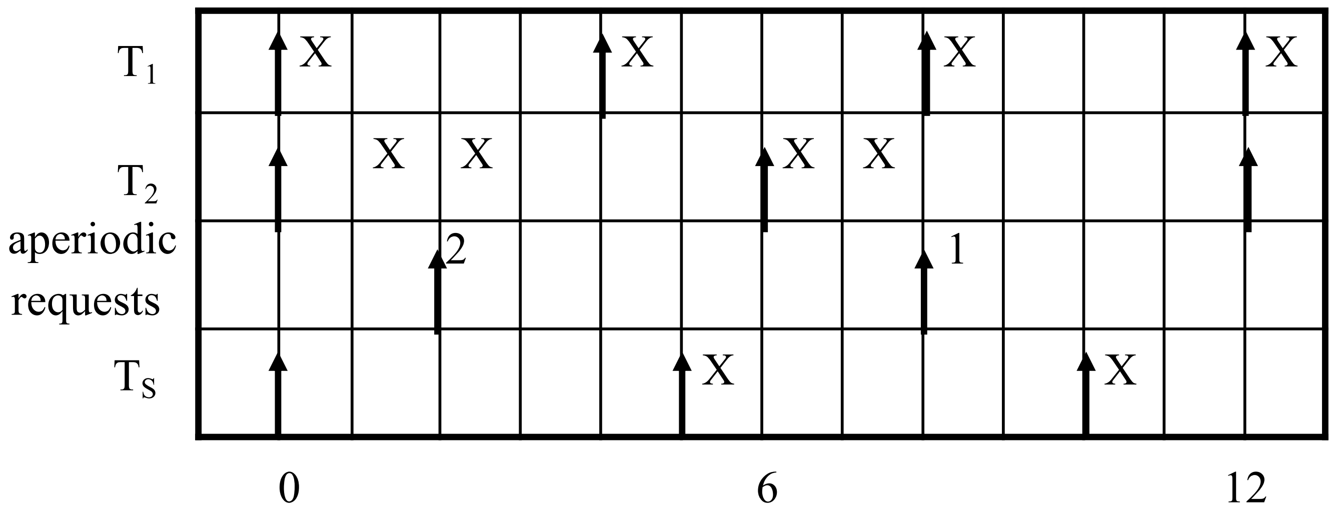
- Schedule the following aperiodic activities by using the polling and sporadic server (without using background)



Class exercise Solution – Polling Server

- Consider the following task set
  - T_1 {C_1=1, p_1= 4}
  - T_2 {C_2=2, p_2= 6}
  - T_s {C_s=1, p_s= 5}

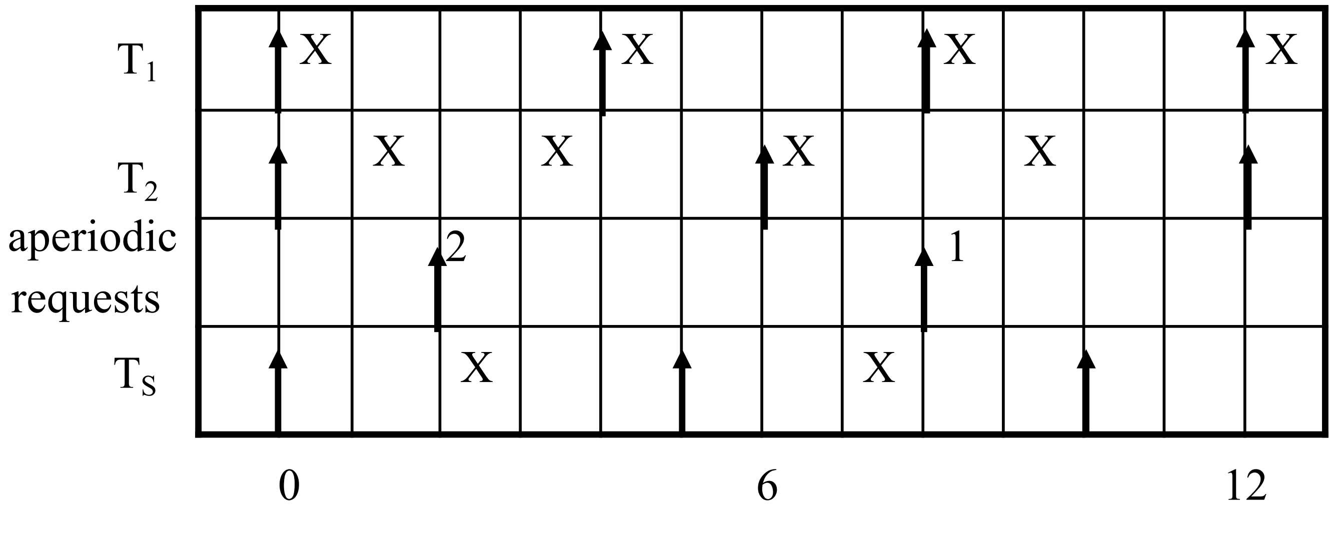
- Schedule the following aperiodic activities by using the polling server (without using background)



Class exercise Solution – Sporadic Server

- Consider the following task set
  - T_1 {C_1=1, p_1= 4}
  - T_2 {C_2=2, p_2= 6}
  - T_s {C_s=1, p_s= 5}

- Schedule the following aperiodic activities by using the sporadic server (without using background)



Class exercise

- Consider the following task set
  - T_1 {C_1=1, p_1= 4}
  - T_2 {C_2=2, p_2= 6}
  - T_s {C_s=1, p_s= 5}

- Are the periodic task set and the Polling Server T_s schedulable?

Class exercise

- Consider the following task set
  - T_1 {C_1=1, p_1= 4}
  - T_2 {C_2=2, p_2= 6}
  - T_s {C_s=1, p_s= 5}

- Are the periodic task set and the Polling Server T_s schedulable?

- T_1 is schedulable because U_1 = ¼ < 1

- Check schedulability of T_1, T_s ➔ OK! $$\frac{C_1}{P_1}+\frac{C_s}{P_s}=0.45<U(2)$$

- Check schedulability of T_1, T_s, T_2 ➔ FAILED! $$\frac{C_1}{P_1}+\frac{C_s}{P_s}+\frac{C_2}{P_2}=0.784>U(3)=0.779$$

- Use exact analysis to check schedulability of T_2

Class exercise

- Consider the following task set
  - T_1 {C_1=1, p_1= 4}
  - T_2 {C_2=2, p_2= 6}
  - T_s {C_s=1, p_s= 5}

- Are the periodic task set and the Polling Server T_s schedulable?
- Use exact analysis to check schedulability of T_2

Class exercise

- Consider the following task set
  - T_1 {C_1=1, p_1= 4}
  - T_2 {C_2=2, p_2= 6}
  - T_s {C_s=1, p_s= 5}

- Are the periodic task set and the Polling Server T_s schedulable?
- Use exact analysis to check schedulability of T_2

$\mathbf{r_2^0 =}$ 1+1+2 = 4,

$\mathbf{r_2^1}$ = ceil(4/4)*1 + ceil(4/5)*1 + 2 = 4 < 6

The lowest priority task T_2 is schedulable!!

Hence, the all task set is schedulable!