

# Scheduling Periodic and Mixed Task Sets

Total Bandwidth Server (TBS), Rate Monotonic (RM), Polling Server

Exercise class 6

---

*Presenter:*

Jürgen Mattheis

*In cooperation with:*  
Pascal Walter

*Based on the lecture of:*  
Marco Zimmerling

*December 20, 2022*

University of Freiburg, Chair for Embedded Systems



# Gliederung

Organisation

Overview

Task 1

Task 2

Task 3

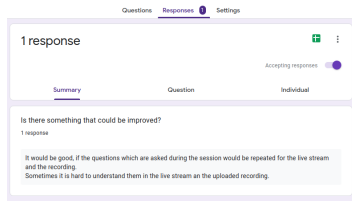
Appendix

Literature

# Organisation

# Organisation I

- ▶ feedback for me: <https://forms.gle/f3YN8EFrZ1vsfPoC6>



The screenshot shows a Google Forms interface with three tabs: 'Questions', 'Responses' (selected), and 'Settings'. Under the 'Responses' tab, it says '1 response'. There are three sub-tabs: 'Summary' (selected), 'Question', and 'Individual'. The 'Summary' sub-tab shows a question: 'Is there something that could be improved?' with '1 response'. The response text is: 'It would be good, if the questions which are asked during the session would be repeated for the live stream and the recording. Sometimes it is hard to understand them in the live stream an the uploaded recording.'

- ▶ get the slides before the exercise class: [https://github.com/matthejue/Einfuehrung\\_in\\_ESE\\_Tutoratsfolien\\_out](https://github.com/matthejue/Einfuehrung_in_ESE_Tutoratsfolien_out)
- ▶ warning: the slides often get changed just shortly before or after the exercise class. Both the lecture and the exercise classes are pretty running edge

# Organisation II

- A** *A (offline)* 1:47 PM

Can one task start in one core be interrupted and resume later in another core?
- A** *A (offline)* 1:55 PM

Thanks
- Ne** *Negin* 1:57 PM

so can one specific task be executed in two cores?

# Organisation III

## Error in sample solution of task 4.2 of sheet 3

Hello Students,

In the english exercise a error in the sample solution was found by coincidence in exercise 4.2 of sheet 3. Some of the summands in the numerator of the sample solution were correct, but some were wrong. We just wanted to inform you that we updated the exercise class slides accordingly, but in the recordings the wrong solution can still be seen

Best regards and have a nice day,

Pascal and Jürgen

# Overview

# Overview Scheduling I

Aperiodic Task Scheduling		
<ul style="list-style-type: none"> <li>event-driven</li> <li>can arrive at any point in time</li> <li><b>Optimality:</b> Minimize the maximum lateness of the task set</li> <li>precedence relations between tasks can be described through an acyclic directed graph <math>G</math> where tasks are represented by nodes and precedence relations by arrows. <math>G</math> induces a partial order on the task set</li> <li><b>2 types of aperiodic tasks:</b> <ul style="list-style-type: none"> <li><b>sporadic task:</b> has a minimum inter-arrival time between consecutive instances of the task</li> <li><b>firm task:</b> maximum inter-arrival time between consecutive instances of the task cannot be bounded</li> </ul> </li> </ul>		
	equal arrival times, non-preemptive	arbitrary arrival times, preemptive
independent tasks	<b>Earliest Deadline Due (EDD)</b> <ul style="list-style-type: none"> <li>priority determined by <math>\min(D_i)</math> for all remaining <math>J_i</math></li> <li><u>Schedulability Test:</u>  <math display="block">\sum_{k=1}^i C_k \leq d_i</math>                     for each task <math>J_i</math> </li> </ul>	<b>Earliest Deadline First (EDF)</b> <ul style="list-style-type: none"> <li>priority determined by <math>\min(d_i)</math> for all remaining <math>J_i</math> that have already arrived (are ready) and not finished every time the arrival time of a task is reached</li> <li><u>Schedulability Test:</u>  <math display="block">t + \sum_{k=1}^i c_k(t) \leq d_i</math>                     for all active tasks <math>J_i</math> <ul style="list-style-type: none"> <li><math>c_k(t)</math> is the remaining worst-case execution time of task <math>J_k</math></li> </ul> </li> </ul>
dependent tasks	<b>Latest Deadline First (LDF)</b> <ul style="list-style-type: none"> <li>priority determined by <math>\max(D_i)</math> for all <math>J_i</math> without successors or whose successors have been all selected in the precedence graph inserted into the queue to be executed last</li> <li>at runtime, tasks are extracted from the head of the queue; the first task inserted in the queue will be executed last</li> </ul>	<b>Earliest Deadline First - Star (EDF*)</b> <ul style="list-style-type: none"> <li>release time and deadline of individual tasks are modified such that all the precedence constraints are satisfied                             <ul style="list-style-type: none"> <li><math>r_i^* = \max(r_i, \max(r_i^* + C_i : J_i \rightarrow J_j))</math></li> <li><math>d_i^* = \min(d_i, \min(d_j^* - C_j : J_i \rightarrow J_j))</math></li> </ul> </li> <li>scheduling problem is transformed into a problem without precedence constraints, which can then be handled by a "normal" EDF scheduler</li> </ul>



# Overview Scheduling II


Periodic Task Scheduling • Mixed Task Sets (Periodic + Aperiodic)		
	<ul style="list-style-type: none"> <li>• Rate-driven</li> <li>• regularly activated with a constant period</li> <li>• Optimality of fixed-priority assignment: every task not schedulable by another fixed-priority algorithm can also be scheduled by FRR and CSR</li> </ul>	
	$D_i = T_i$	$D_i \leq T_i$
	<b>Rate Monotonic (RM)</b> <ul style="list-style-type: none"> <li>• priority determined by <math>\text{rate}(T_i)</math></li> <li>for all remaining <math>J_i</math></li> <li>• <b>Schedulability Test 1:</b>  <math display="block">U = \sum_{i=1}^n \frac{C_i}{T_i} \leq n \left( 2^{1/n} - 1 \right)</math>                     (sufficient but not necessary)                 </li> <li>• <b>Schedulability Test 2:</b> same as Schedulability Test 1 for CSR</li> </ul>	<b>Deadline Monotonic (DM)</b> <ul style="list-style-type: none"> <li>• priority determined by <math>\text{rate}(D_i)</math></li> <li>for all remaining <math>J_i</math></li> <li>• <b>Schedulability Test 1:</b>  <math display="block">\sum_{i=1}^n \frac{C_i}{D_i} \leq n \left( 2^{1/n} - 1 \right)</math>                     (sufficient but not necessary)                 </li> <li>• <b>Schedulability Test 2:</b> for all tasks <math>\tau_i</math> <math display="block">R_i = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_j}{T_j} \right\rceil C_j</math>                     that satisfies:                     <math display="block">R_i \leq C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_j}{T_j} \right\rceil C_j</math>                     and <math>R_i \leq D_i</math> (necessary and sufficient)                 </li> </ul>
	<b>Background Scheduling (BS)</b> <ul style="list-style-type: none"> <li>• aperiodic tasks scheduled after FRR when no periodic task ready to execute</li> </ul>	
	<b>Putting Server (PS):</b> <ul style="list-style-type: none"> <li>• PS scheduled as periodic task, it serves any pending aperiodic requests until its capacity (execution time) <math>C_p</math> is exhausted</li> <li>• if no aperiodic requests are pending, PS suspends itself until the beginning of its next period, and the budget originally allocated for aperiodic service is freed up and assigned to periodic tasks</li> <li>• <b>Schedulability Test 1:</b>  <math display="block">\sum_{i=1}^n \frac{C_i}{T_i} + \frac{C_p}{T_p} \leq (n+1) \left( 2^{1/(n+1)} - 1 \right)</math>                     (sufficient but not necessary)                 </li> <li>• <b>Schedulability Test 2:</b> <math display="block">C_p \leq C_s</math>                     and <math>2T_p \leq D_p</math>                     (necessary and sufficient, using completion time)                 </li> <li>• <b>Schedulability Test 3:</b>  <math display="block">T_p + \left\lceil \frac{C_p}{C_s} \right\rceil T_s \leq D_p</math>                     (necessary and sufficient, not fitting completion time)                 </li> </ul>	
	<b>Harvest Deadline First (HDF)</b> <ul style="list-style-type: none"> <li>• <b>Schedulability Test:</b>  <math display="block">\sum_{i=1}^n \frac{C_i}{T_i} = U \leq 1</math> </li> </ul>	
	<b>Total Bandwidth Server (TBS)</b> <ul style="list-style-type: none"> <li>• <math display="block">d_k = \max\{r_k, d_{k-1}\} + \frac{C_k}{U_s}</math></li> <li>• <b>Schedulability Test:</b>  <math display="block">U_s + U_r \leq 1</math>                     (necessary and sufficient)                 </li> </ul>	<b>Harvest Deadline First (HDF)</b> <ul style="list-style-type: none"> <li>• <b>Schedulability Test:</b> Necessary. Have the first computing system: pre-emptive scheduling</li> </ul>
static priority		
dynamic priority		

# Overview Scheduling III

### Important Parameters


#### Periodic Tasks

- $\Gamma$  : set of periodic tasks
- $\tau_i$  : periodic task
- $\tau_{i,j}$  :  $j$ -th instance of task  $\tau_i$
- $r_{i,j}, s_{i,j}, f_{i,j}, d_{i,j}$  : release time, start time, finishing time, and absolute deadline of the  $j$ -th instance of task  $\tau_i$
- $\Phi_i$  : phase of task  $\tau_i$  (release time of first instance)
- $C_i$  : worst-case execution time of task  $\tau_i$
- $T_i$  : period of task  $\tau_i$
- $D_i$  : relative deadline of task  $\tau_i$

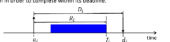


#### Real-Time Tasks

- Arrival time**  $a_i$  or **release time**  $r_i$  is the time at which a task becomes ready for execution.
- Start time**  $s_i$  is the time at which a task starts its execution.
- Execution time**  $C_i$  is the time needed by the processor to execute a task without interruption.
- Finishing time**  $f_i$  is the time at which a task finishes its execution.
- Absolute deadline**  $d_i$  is the time by which a task should be completed.




- Relative deadline**  $D_i$  is the difference between the absolute deadline and the arrival time of a task, that is,  $D_i = d_i - a_i$ .
- Response time**  $R_i$  is the difference between the finishing time and the arrival time of a task, that is,  $R_i = f_i - a_i$ .
- Lateness**  $L_i = f_i - d_i$  represents the delay of a task completion with respect to its deadline, that is,  $L_i \leq 0$  if a task completes within its deadline.
- Forfeiture**  $F_i = \max(0, L_i)$  is the time a task stays active after its deadline.
- Slack**  $S_i = d_i - a_i - C_i$  is the maximum time a task can be delayed on its execution in order to complete within its deadline.



#### Critical Instant

- Definition:** A critical instant of a task is the release time  $r_i$  that produces the largest response time  $R_i$ , that is, the largest difference between release time  $r_i$  and finishing time  $f_i$ .
- Lemma:** For any task, a critical instant occurs whenever the task is released simultaneously with all higher-priority tasks.



# Overview Scheduling IV

Important Formulas	
<ul style="list-style-type: none"> <li>release time:</li> </ul>	$r_{i,j} = \Phi_i + (j-1)T_i$
<ul style="list-style-type: none"> <li>absolute deadline:</li> </ul>	$d_{i,j} = r_{i,j} + D_i = \Phi_i + (j-1)T_i + D_i$
and if	
	$D_i = T_i$
then	
	$d_{i,j} = \Phi_i + jT_i$
<ul style="list-style-type: none"> <li>rate:</li> </ul>	$\frac{1}{T_i}$
<ul style="list-style-type: none"> <li>processor utilization:</li> </ul>	$U_p = \sum_{i=1}^n \frac{C_i}{T_i}$
Metrics to compare schedules	
<ul style="list-style-type: none"> <li>Average response time: <math>\bar{r}_r = \frac{1}{n} \sum_{i=1}^n (f_i - a_i)</math></li> </ul>	
<ul style="list-style-type: none"> <li>Total completion time: <math>t_c = \max_i(f_i) - \min_i(a_i)</math></li> </ul>	
<ul style="list-style-type: none"> <li>Weighted sum of response times: <math>t_w = \frac{\sum_{i=1}^n w_i (f_i - a_i)}{\sum_{i=1}^n w_i}</math></li> </ul>	
<ul style="list-style-type: none"> <li>Maximum lateness: <math>L_{\max} = \max_i(f_i - d_i)</math></li> </ul>	
<ul style="list-style-type: none"> <li>Number of late tasks: <math>N_{\text{late}} = \sum_{i=1}^n \text{miss}(f_i)</math></li> </ul>	
where $\text{miss}(f_i) = \begin{cases} 0 & \text{if } f_i \leq d_i \\ 1 & \text{otherwise} \end{cases}$	Only these metrics are useful to evaluate real-time schedules, as they involve task <b>deadlines</b> .

# Task 1

# Task 1 I

## Earliest Deadline First (EDF) and Total Bandwidth Server (TBS)

### Task 1.1:

	$\tau_1$	$\tau_2$	$\tau_3$
$C_i$	1	1	2
$T_i$	3	5	13

- ▶ TBS executes along with the **periodic tasks** above
- ▶ what can be the maximum value of  $U_s$  such that the whole set (i.e. periodic tasks and the **TBS**) is schedulable with **EDF**?

# Task 1 II

## Earliest Deadline First (EDF) and Total Bandwidth Server (TBS)

### Requirements 1.1:

#### *Schedulability test:*

Given a set of  $n$  periodic tasks with processor utilization  $U_p$  and a total bandwidth server with utilization  $U_s$ , the whole set is schedulable by EDF if and only if

$$U_p + U_s \leq 1$$

► *processor utilization factor:*  $U = \sum_{i=1}^n \frac{C_i}{T_i}$

# Task 1 III

## Earliest Deadline First (EDF) and Total Bandwidth Server (TBS)

### Solution 1.1:

- ▶ *Maximum utilization of the Total Bandwidth Server:*

$$U_{s,\max} = 1 - U_p = 1 - \left(\frac{1}{3} + \frac{1}{5} + \frac{2}{13}\right) = \frac{61}{195} \approx 0.3128$$

# Task 1 I

## Earliest Deadline First (EDF) and Total Bandwidth Server (TBS)

### Task 1.2:

- ▶ construct EDF-Schedule
- ▶ assume  $U_s = 0.25$
- ▶ three aperiodic requests served by TBS:

	$J_4$	$J_5$	$J_6$
$r_i$	0	15	10
$C_i$	2	1	1

- ▶ arrival time of first instance of each periodic task is 0:

	$\tau_1$	$\tau_2$	$\tau_3$
$a_i$	0	0	0
$C_i$	1	1	2
$T_i$	3	5	13



# Task 1 II

## Earliest Deadline First (EDF) and Total Bandwidth Server (TBS)

### Requirements 1.2:

- ▶  $d_i = \max(r_i, d_{k-1}) + \frac{C_k}{U_s}$ 
  - ▶  $d_{k-1}$  denotes the previously calculated deadline ( $k - 1$  means the predecessor in the ordering according to the release time)

### Solution 1.2:

- ▶ order the aperiodic tasks by **increasing release time**  $r_i : J_4, J_6, J_5$
- ▶ calculate the deadlines:
  - ▶  $d_4 = \max(r_4, d_0) + \frac{2}{0.25} = \max(0, 0) + 8 = 8$
  - ▶  $d_6 = \max(r_6, d_4) + \frac{1}{0.25} = \max(10, 8) + 4 = 14$
  - ▶  $d_5 = \max(r_5, d_6) + \frac{1}{0.25} = \max(15, 14) + 4 = 19$

# Task 1 III

## Earliest Deadline First (EDF) and Total Bandwidth Server (TBS)

Solution 1.2:

	$\tau_1$	$\tau_2$	$\tau_3$
$a_i$	0	0	0
$C_i$	1	1	2
$T_i$	3	5	13
	$J_4$	$J_5$	$J_6$
$r_i$	0	15	10
$d_i$	8	19	14
$C_i$	2	1	1

# Task 1 IV

## Earliest Deadline First (EDF) and Total Bandwidth Server (TBS)

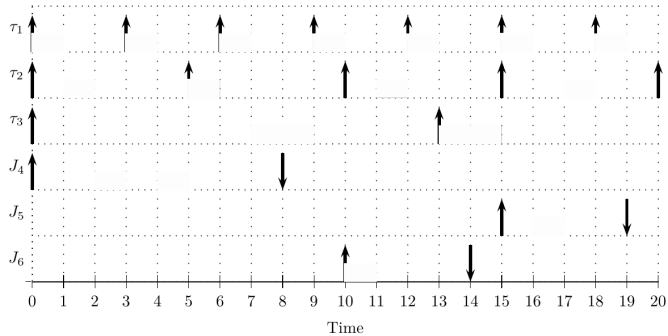


Figure 1: EDF schedule solution for Task 1

# Task 1 V

## Earliest Deadline First (EDF) and Total Bandwidth Server (TBS)

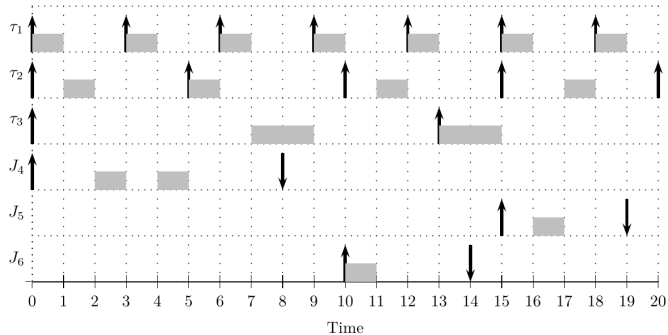


Figure 1: EDF schedule solution for Task 1

# Task 2

# Task 2 I

## Schedulability Test for Fixed Priorities – Rate Monotonic (RM)

### Task 2.1:

- ▶ task-set schedulable with **RM**
- ▶ using **sufficient** test

	$\tau_1$	$\tau_2$	$\tau_3$
$C_i$	1	3	2
$T_i$	3	8	9

### Requirements 2.1:

- ▶  $U = \sum_{i=1}^n \frac{C_i}{T_i} \leq n \left( 2^{1/n} - 1 \right)$ ,  $U$  is the **fraction** of the **processor time** spent on **executing task set**

## Task 2 II

### Schedulability Test for Fixed Priorities – Rate Monotonic (RM)

#### Solution 2.1:

- ▶  $\frac{1}{3} + \frac{3}{8} + \frac{2}{9} = 0.93 \leq 3(2^{\frac{1}{3}} - 1) = 0.78 \quad \times$
- ▶ condition is *not necessary*, hence we *don't know* whether the task set is schedulable with *RM* or not

# Task 2 III

## Schedulability Test for Fixed Priorities – Rate Monotonic (RM)

### Task 2.2:

- ▶ task-set schedulable with RM
- ▶ using necessary test

	$\tau_1$	$\tau_2$	$\tau_3$
$C_i$	1	3	2
$T_i$	3	8	9



# Task 2 IV

## Schedulability Test for Fixed Priorities – Rate Monotonic (RM)

### Requirements 2.2:

- ▶ guarantee that *all* the tasks can be scheduled in *any possible instance*
- ▶ in particular, if a task can be scheduled in its *critical instances*, then the schedulability guarantee condition holds
  - ▶ a *critical instance* of a task occurs whenever the task is *released simultaneously* with all higher priority tasks
- ▶ **Schedulability Test:** For all tasks  $\tau_i$  smallest  $R_i$  that satisfies

$$R_i = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \text{ and } R_i \leq D_i \text{ (necessary and sufficient)}$$

# Task 2 V

## Schedulability Test for Fixed Priorities – Rate Monotonic (RM)

### Requirements 2.2:

```

DM_guarantee ( $\Gamma$ ) {
  for (each  $\tau_i \in \Gamma$ ) {
     $I_i = \sum_{k=1}^{i-1} C_k$ ;
    do {
       $R_i = I_i + C_i$ ;
      if ( $R_i > D_i$ ) return(UNSCHEDULABLE);
       $I_i = \sum_{k=1}^{i-1} \left\lceil \frac{R_i}{T_k} \right\rceil C_k$ ;
    } while ( $I_i + C_i > R_i$ );
  }
  return(SCHEDULABLE);
}
    
```

# Task 2 VI

## Schedulability Test for Fixed Priorities – Rate Monotonic (RM)

### Solution 2.2:

- ▶ *The tasks are first ordered by their priorities:  $\tau_1$ ,  $\tau_2$  and  $\tau_3$* 
  - ▶ *In this case the tasks are already ordered*
- ▶  $\tau_3$  :
  - ▶  $R_3^0 = 1 + 3 + 2 = 6 \quad I_3^0 = \left\lceil \frac{2}{3} \right\rceil 1 + \left\lceil \frac{2}{8} \right\rceil 3 = 1 + 3 = 4 \quad 4 + 6 \neq 2$
  - ▶  $R_3^1 = 4 + 2 = 6 \quad I_3^1 = \left\lceil \frac{6}{3} \right\rceil 1 + \left\lceil \frac{6}{8} \right\rceil 3 = 2 + 3 = 5 \quad 5 + 2 \neq 6$
  - ▶  $R_3^2 = 5 + 2 = 7 \quad I_3^2 = \left\lceil \frac{7}{3} \right\rceil 1 + \left\lceil \frac{7}{8} \right\rceil 3 = 3 + 3 = 6 \quad 6 + 2 \neq 7$
  - ▶  $R_3^3 = 6 + 2 = 8 \quad I_3^3 = \left\lceil \frac{8}{3} \right\rceil 1 + \left\lceil \frac{8}{8} \right\rceil 3 = 3 + 3 = 6 \quad 6 + 2 = 8 \dots \checkmark$   
 (since  $R_3 = 8 \leq T_3 = 9$ )

# Task 2 VII

## Schedulability Test for Fixed Priorities – Rate Monotonic (RM)

### Solution 2.2:



▶  $\tau_2$ :

▶  $R_2^0 = 1 + 3 \quad I_2^0 = \lceil \frac{3}{3} \rceil 1 = 1 \quad 1 + 4 \neq 3$

▶  $R_2^1 = 1 + 3 = 4 \quad I_2^1 = \lceil \frac{4}{3} \rceil 1 = 2 \quad 2 + 3 \neq 4$

▶  $R_2^2 = 2 + 3 = 5 \quad I_2^2 = \lceil \frac{5}{3} \rceil 1 = 2 \quad 2 + 3 = 5 \dots \checkmark$

(since  $R_2 = 5 \leq T_2 = 8$ )

▶  $\tau_1$ :

▶  $R_1^0 = C_1 = 1 \quad I_1^0 = 0 \quad 0 + 1 = 1 \dots \checkmark$

(since  $R_1 = 1 \leq T_1 = 3$ )

▶ *The necessary and sufficient test succeeds. This means that the task set is schedulable with RM.*

# Task 2 VIII

## Schedulability Test for Fixed Priorities – Rate Monotonic (RM)

### Task 2.3:

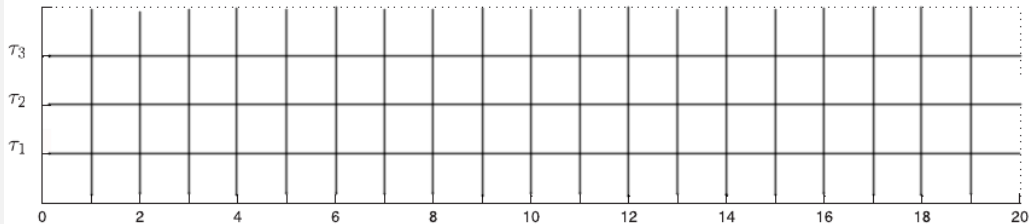
- ▶ first job of each task arrives at time 0
- ▶ construct schedule for interval  $[0, 20]$

	$\tau_1$	$\tau_2$	$\tau_3$
$r_i$	0	0	0
$C_i$	1	3	2
$T_i$	3	8	9

# Task 2 IX

## Schedulability Test for Fixed Priorities – Rate Monotonic (RM)

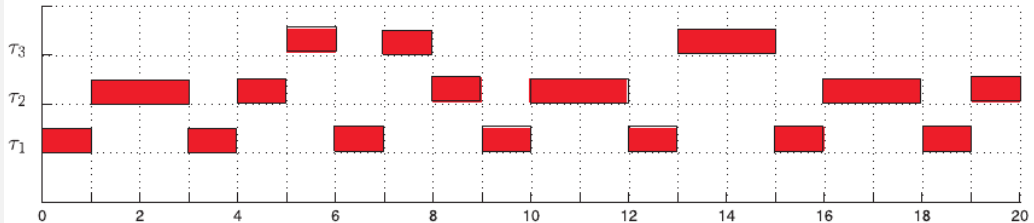
### Solution 2.3:



# Task 2 X

## Schedulability Test for Fixed Priorities – Rate Monotonic (RM)

### Solution 2.3:



# Task 3



# Task 3

## Scheduling with Polling Server

### Task 3.1:



	$\tau_1$	$\tau_2$	$\tau_3$
$C_i$	2	2	2
$D_i$	6	8	16
$T_i$	6	8	16

- ▶ In addition to the above periodic tasks, we have an aperiodic job  $J_a$  with  $C_a = 1$  and relative deadline  $D_a$ . Let  $T_s = 25$  and  $C_s = 1$  respectively, where  $T_s$  denotes the period and  $C_s$  the computing time (or capacity) of the polling server (PS)
- ▶ Compute the minimum relative deadline of  $J_a$  which is guaranteed not to be missed, that is, its aperiodic guarantee.

# Task 3 I

## Scheduling with Polling Server

### Solution 3.1:

- ▶ *As reminder: The polling server itself acts like a **periodic** task, that uses its capacity to serve **aperiodic** tasks. If the polling server has the current highest priority it begins to serve any pending aperiodic requests within the limits of its capacity.*
- ▶ *If there are no pending aperiodic tasks at that time it **suspends** its entire capacity! (until the beginning of the next period)*
- ▶ *Therefore, the worst possible case occurs when  $J_a$  appears slightly later than the polling server checks for pending aperiodic tasks. Meaning, the polling server suspends its capacity and  $J_a$  has to wait  $T_s + \lceil \frac{C_a}{C_s} \rceil T_s = (1 + \lceil \frac{C_a}{C_s} \rceil) T_s$*

# Task 3 II

## Scheduling with Polling Server

### Solution 3.1:



- ▶ To guarantee to not miss the deadline  $D_a$  the condition  $(1 + \lceil \frac{C_a}{C_s} \rceil) T_s \leq D_a$  needs to hold.
- ▶ Entering the values for the exercise gives us  $D_a = (1 + \frac{1}{1}) \cdot 25 = 50$

### Sidenote 🔍

Note that the above computation of course only holds if the RM schedule meets all the deadlines.

# Task 3

## Scheduling with Polling Server

### Task 3.2:

Using the sufficient test of RM, test if the polling server of task 3.1 is schedulable along with the periodic task-set.

### Requirements 3.2:

► *Schedulability Test:* 
$$\sum_{i=1}^n \frac{C_i}{T_i} + \frac{C_s}{T_s} \leq (n+1) \left[ 2^{1/(n+1)} - 1 \right]$$

# Task 3

## Scheduling with Polling Server

### Solution 3.2:

- ▶ We have already seen the sufficient but not necessary condition of  $\sum_{i=1}^n \frac{C_i}{T_i} \leq n(2^{1/n} - 1)$  for rate monotonic scheduling.
- ▶ The convenient part about our polling server: we can just treat it as an additional *periodic* task.
- ▶ Therefore, the same test offers us a sufficient condition for rate monotonic scheduling with a polling server! Simply increase from  $n$  tasks to  $(n+1)$
- ▶ Check:  $\sum_{i=1}^n \frac{C_i}{T_i} + \underbrace{\frac{C_s}{T_s}}_{\text{server task}} \leq (n+1)(2^{1/(n+1)} - 1)$

# Task 3

## Scheduling with Polling Server

### Solution 3.2:

► *Putting in values we obtain:*

$$\frac{2}{6} + \frac{2}{8} + \frac{2}{16} + \frac{1}{25} \leq (3 + 1)(2^{\frac{1}{3+1}} - 1) \Leftrightarrow \frac{449}{600} \leq \sqrt[4]{256} \cdot \sqrt[4]{2} - 4 \Leftrightarrow 0.75 \leq 0.76.$$

*Since this is true, we know (as this is a sufficient condition) that the RM schedule meets all deadlines!*

# Appendix

# Overview periodic Task Scheduling

	Deadline equals period	Deadline smaller than period
<b>static priority</b>	RM (rate-monotonic)	DM (deadline-monotonic)
<b>dynamic priority</b>	EDF	EDF



# Overview Aperiodic Task Scheduling

## Schedulability test

	Deadline equals period ( $D_i = T_i$ )	Deadline smaller than period ( $D_i \leq T_i$ )
static priority	<p>(1) <math>\sum_{i=1}^n \frac{C_i}{T_i} \leq n \left( 2^{1/n} - 1 \right)</math>                      (sufficient but not necessary)                      (2) same as Schedulability Test 2 for DM</p>	<p>(1) <math>\sum_{i=1}^n \frac{C_i}{D_i} \leq n \left( 2^{1/n} - 1 \right)</math>                      (sufficient but not necessary)                      (2) smallest <math>R_i</math> that satisfies  <math display="block">R_i = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_j}{T_j} \right\rceil C_j</math>                     and <math>R_i \leq D_i</math> for all tasks <math>\tau_i</math>                      (necessary and sufficient)</p>
dynamic priority	<p><math>\sum_{i=1}^n \frac{C_i}{T_i} = U \leq 1</math>                      (necessary and sufficient)</p>	<p>→ Buttazzo, <i>Hard real-time computing systems: predictable scheduling algorithms and applications</i></p>

# Mixed Task Sets

- ▶ So far: we differentiated between **periodic** and **aperiodic** tasks.
- ▶ Now: Consider a **mixed** task set!
- ▶ We want to be able to find a schedule when there's both **periodic** and **aperiodic** tasks.

# Schedulability tests

## Sufficient? Necessary?

- ▶ We're interested in whether a given problem can be scheduled by algorithms.
- ▶ Depending on the algorithm we can derive sufficient and necessary conditions.

**Sufficient:** If  $A \implies B$  then A is a sufficient condition for B.

**Necessary:** If  $B \implies A$  then A is a necessary condition for B.

- ▶ A necessary and sufficient condition means, both statements are logically equivalent.

# Schedulability tests

## Utilization

Different kind of utilizations also play a big role in our analysis. We introduced the **processor utilization factor**  $U = \sum_{i=1}^n \frac{C_i}{T_i}$  and later on  $U_s$  as the server utilization.

(More about servers later)

# RM - Rate Monotonic Scheduling

## Schedulability

- ▶ RM is optimal among all fixed-priority assignments in the sense that no other fixed-priority algorithm can schedule a task set that cannot be scheduled by RM.
- ▶ As in the lecture, we have  $\sum_{i=1}^n \frac{C_i}{T_i} \leq n(2^{1/n} - 1)$  as a **sufficient** but not **necessary** condition.

# RM(PS) - Rate Monotonic Polling Server

- ▶ One way to handle both periodic and aperiodic tasks is to use a so called server.
- ▶ This PS (Polling Server) acts as a periodic task (meaning it is instantiated at regular intervals  $T_s$ ) whose job it is to, once it has the highest priority, serve any pending aperiodic requests within the limits of a server capacity  $C_s$ .
- ▶ Since we introduce yet another periodic task, the schedulability analysis simply is the same as normal *RM* with one additional task. Again, we have the **sufficient** but not **necessary** condition: 
$$\frac{C_s}{T_s} + \sum_{i=1}^n \frac{C_i}{T_i} \leq (n+1)(2^{1/(n+1)} - 1)$$

# Literature

# Bücher



[Buttazzo, Giorgio C.](#) *Hard real-time computing systems: predictable scheduling algorithms and applications*. Vol. 24. Springer Science & Business Media, 2011.