MEC302: Embedded Computer Systems

Theme II: Design of Embedded Computer Systems

Lecture 9 – Scheduling

Dr. Timur Saifutdinov

Assistant Professor at EEE, SAT

Email: Timur.Saifutdinov@xjtlu.edu.cn

Outline

- Basics of scheduling:
  - Implementation of a scheduler;
  - Scheduling decisions;
  - Task model;
  - Scheduler metrics;
- Scheduling strategies:
  - Single processor scheduling;
  - Multiprocessor scheduling;
- Scheduling anomalies.

Basics of scheduling

Scheduling is one of the most central functions of an OS (micro)kernel that affects performance of the computer system.

Scheduler is an OS procedure that decides what task(s) to execute next when there is a choice in the execution of a concurrent program or a set of programs. They might be:

- **Non-preemptive scheduler** – always waits for a task to finish before assigning another one;
- **Preemptive scheduler** – may interrupt a task and assign another one for execution when, e.g.:
    - A **timer interrupt** occurs (e.g., **jiffy**);
    - An **I/O interrupt** occurs;
    - An **OS service** is invoked;
    - A task attempts acquire a **mutex**.

Implementation of a scheduler may be provided by:

- A **compiler** or **code generator** – scheduling decisions are made at program **design time**;
- **Operating system (micro)kernel** – scheduling decisions are made at **run time**.
- Both – some decisions are made at **design time** and some at **run time**.

Basics of scheduling

A **scheduling** includes three **decisions**:

- **Assignment** – which processor should execute the task;
- **Ordering** – in what order each processor should execute its tasks;
- **Timing** – the time at which each task executes.

#Depending on when the **decisions** are made (**design time** or **run time**), there are different types of schedulers:

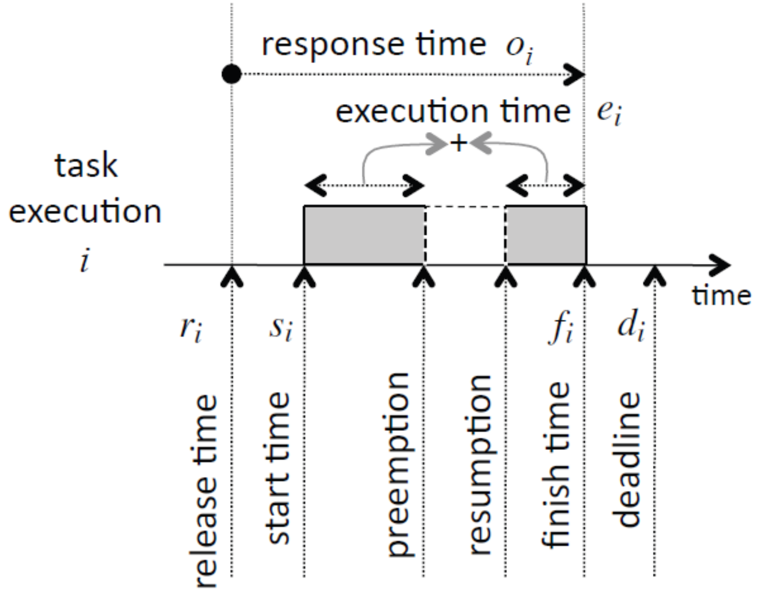
- Fully-static scheduler – makes all decisions at **design time**;
- Static order scheduler – **assignment** and **ordering** at **design time**, **timing** at **run time*;***
- Static assignment scheduler – **assignment** at **design time**, **ordering** and **timing** at **run time**;
- Fully-dynamic scheduler – all decisions are performed at **run time**.

Task model of the scheduler

Task model is the set of assumptions on task(s) used by the scheduler to make a decision:

- **Arrival of tasks** – all tasks known, periodic or sporadic;
- **Precedence constraints of the task(s)** – one execution of a task should precede another (e.g., order);
- **Preconditions of a task** – enable a task for execution (e.g., mutex lock availability);
- **Priority** – measure of importance of tasks;
- **Timeline of a task**:
  - **Release time** (r_i) – earliest time at which a task is enabled;
  - **Start time** (s_i) – execution actually starts;
  - **Preemption** – execution was interrupted;
  - **Resumption** – execution was resumed;
  - **Finish time** (f_i) – task completes execution;
  - **Deadline** (d_i) – time by which the task must be completed;

Timeline of a task execution i:



- **Response time** (o_i) – time duration between release and finish times;
$$o_i = f_i - r_i$$
- **Execution time** (e_i) – total time that the task is actually executing.

Scheduler metrics

There is no single metric to assess and/or compare schedulers.

Usually, the goal of a scheduler is to have a **feasible schedule** – all task executions meet their **deadlines** (i.e., **optimal with respect to feasibility**):

$$f_i \leq d_i \; \forall \, i \in T.$$

Scheduler effectiveness can be measured quantitatively with respect to:

- **Processor utilization** – percentage of time processor executes tasks (vs. total completion time):

$$\mu = \frac{\sum_{i \in T} e_i}{\max_{\mathrm{i} \in T} f_i - \min_{i \in T} r_i};$$

- **Maximum lateness** – can be either positive and non-positive (infeasible and feasible schedules, respectively):

$$L_{max} = \max_{i \in T}(f_i - d_i);$$

- **Makespan** – total completion time:

$$M = \max_{i \in T} f_i - \min_{i \in T} r_i .$$

Scheduling strategies

Main requirements for a **scheduler**:

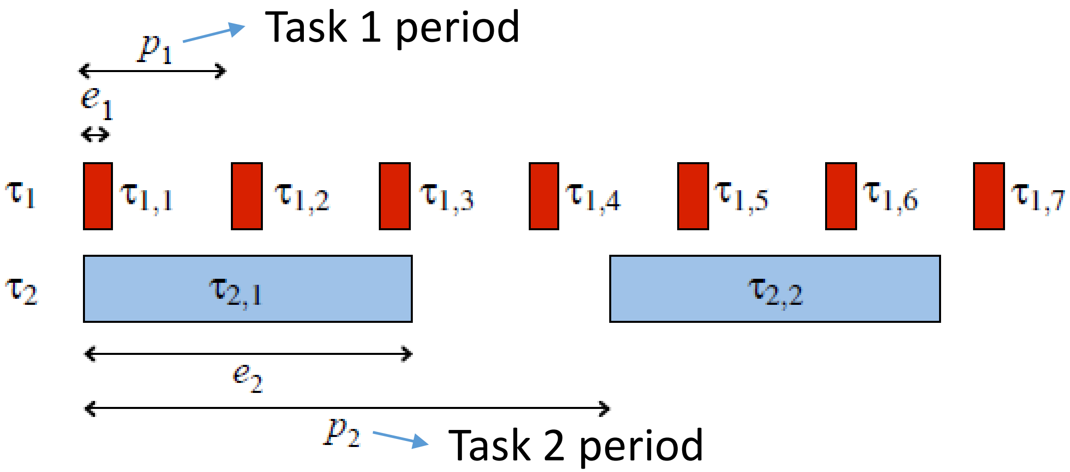
- **Simple** – not require significant resources (i.e., memory and computation);
- **Effective** – do its job (e.g. minimizes maximum lateness or provide feasible schedule);
- **Robust** – respond to various types of tasks (i.e., task models);

!The choice of a scheduling strategy is governed by the goals of the application.

#Consider a scenario with $T = \{\tau_1, \tau_2, \dots, \tau_n\}$ of n tasks and their periods $P = \{p_1, p_2, \dots, p_n\}$:

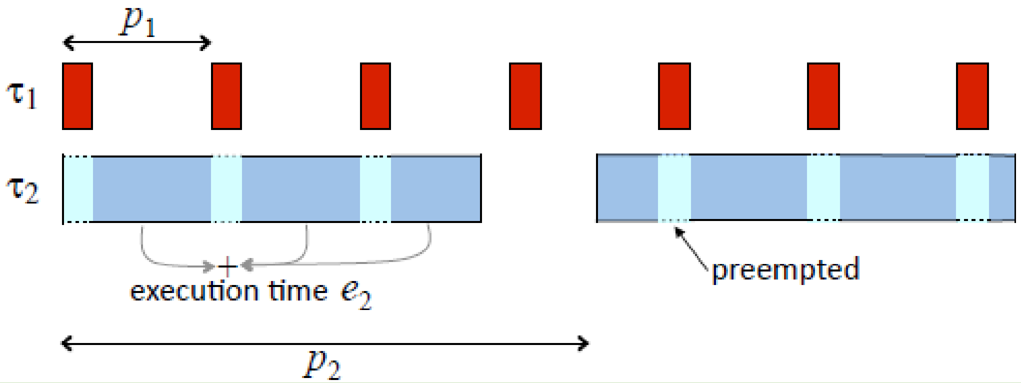
- The simplest strategy to apply is fixed priority preemptive scheduling (i.e., **Rate Monotonic (RM) scheduling**), where a higher priority is given to a task with smaller period.

#The simplest scenario with two tasks:



- Since $e_2 \geq p_1$, a **non-preemptive** scheduler will not be feasible.

Preemptive RM strategy is feasible :



Th1: If any fixed priority ordering yields a feasible schedule, then **RM strategy** always yields a feasible schedule.

Scheduling strategies

1. RM scheduler is easy to implement using:

- Timer interrupt with greatest common divisor of the periods of the tasks or multiple timer interrupts;
- Fixed priorities to schedule the tasks (i.e., based on the duration of periods of the tasks – from low to high).

!There are situations (e.g., tasks timelines) when **RM** cannot achieve 100% processor utilization:

- There are idle processor cycles that cannot be used without causing infeasibility (i.e., missing deadlines).

In case of periodic tasks, processor utilization is:

$$\mu = \sum_{t \in T} \frac{e_i}{p_i}$$

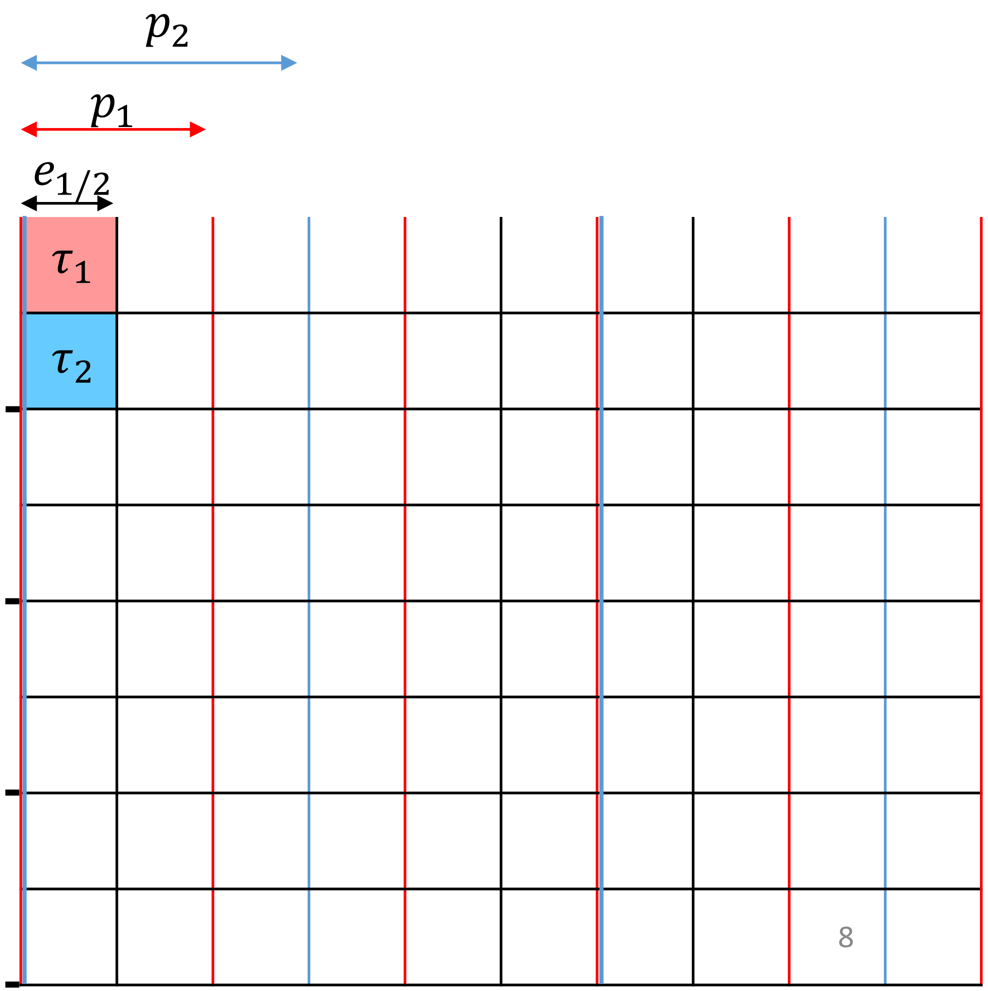
#Consider the two task scenario with:

•Execution times $e_1 = e_2 = 1$;

•Task periods $p_1 = 2$ and $p_2 = 3$:

I. Try scheduling according RM;

II. Do the same for either e_1 or $e_2 > 1$.



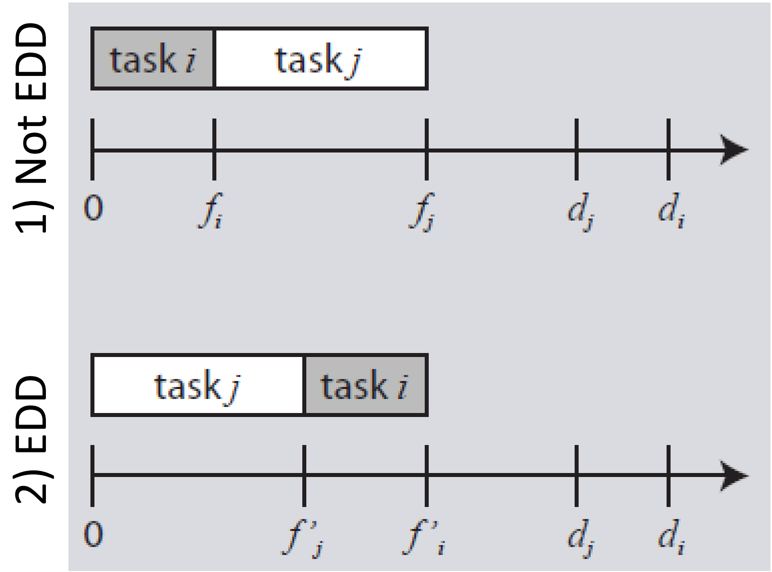
Scheduling strategies

2. For a finite set of non-repeating tasks and no precedence constraints a simple but yet effective strategy is **Earliest Due Date (EDD)**:

- The **EDD** strategy executes the tasks in the order as their deadlines (i.e., earliest deadline first);
- If two or more tasks have the same deadline, then their relative order does not matter;

Th2:
- For a finite set of non-repeating tasks without precedence, EDD minimizes the maximum lateness:
$$L_{max} = \max_{i \in T}(f_i - d_i)\;;$$
- Since **EDD** minimizes the maximum lateness, it is optimal with respect to feasibility (as long as there exists a feasible schedule).
- **EDD** does not support arrival of tasks and precedence.

#Consider two tasks j and i with deadlines $d_j < d_i$, which can be sch'd:



We need to show that $L'_{\max} \leq L_{\max}$:

1) $L_{\max} = \max\left(f_i - d_i, f_j - d_j\right) = f_j - d_j$

2) $L'_{max} = \max\left(f_i' - d_i, f_j' - d_j\right)$

There are two possibilities. Either,

i) $L'_{max} = f_i' - d_i < f_j - d_j$ as $f_i' = f_j$;

ii) $L'_{max} = f_j' - d_j < f_j - d_j$ as $f_j' < f_j$

QED

Scheduling strategies

3. For a set of tasks T without precedence arriving arbitrary, a simple and effective strategy is **Earliest Deadline First(EDF)**:

- **EDF** is a preemptive dynamic priority scheduling strategy that at any instant of time executes the task with the earliest deadline among all arrived tasks;
- **EDF** minimizes the maximum lateness (even for unbounded set of tasks).
- Not effective if there are precedence.

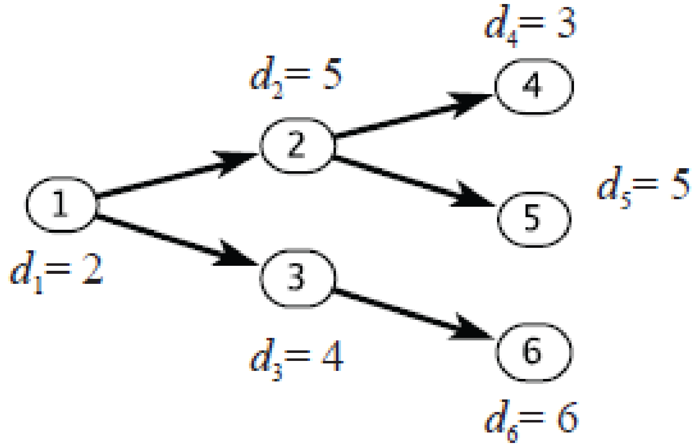
4. EDF with Precedences (EDF*):

- **EDF*** follows **EDF** strategy after modifying deadlines:

$$d_i' = \min\left(d_i,\ \min_{j \in D(i)} (d_j' - e_j)\right),$$

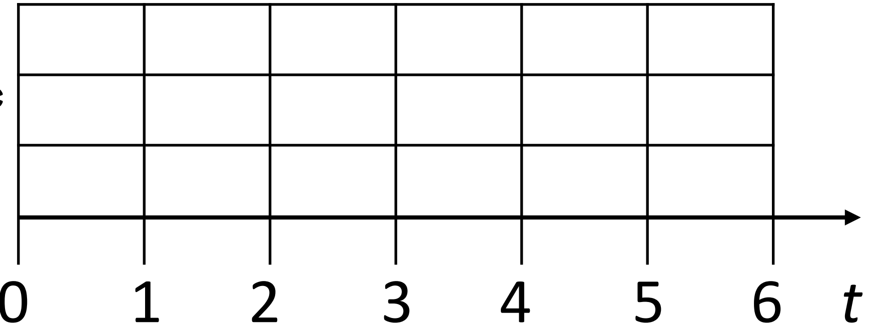
where $D(i) \subset T$ is the set of tasks that immediately depend on task i.

#Consider an example of tasks with precedence ($e_n = 1$):



Now, let's schedule tasks with:

3.EDF
4.EDF*
5.LDF



5. Much simpler algorithm for a finite set of tasks(no arrivals)with precedence is **Latest Deadline First (LDF)**:

- **LDF** schedules tasks backwards, choosing the one with the latest deadline first.

Multiprocessor scheduling

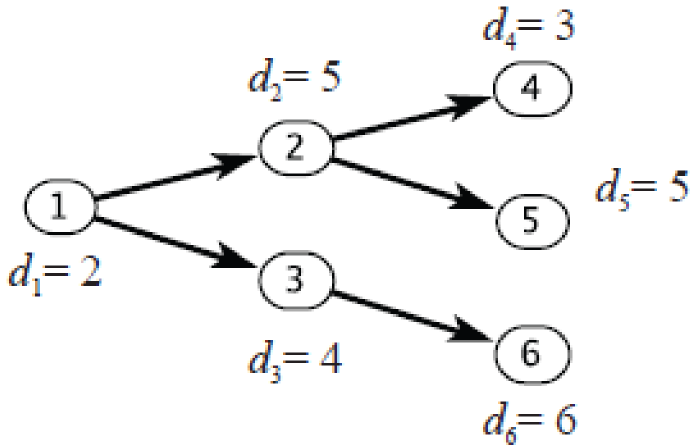
Multiprocessor scheduling requires **assigning** tasks to processors, making it an even harder problem; however, effective strategies exist!

6. For a finite set of non-repeating tasks with precedence constraints a simple but yet effective multicore strategy is **Hu Level Scheduling (HLS)**:

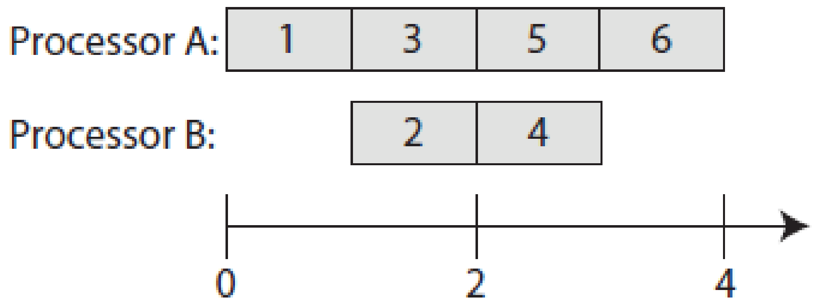
- **HLS** is a <u>priority-based</u> strategy for multiple cores;
- **HLS** assigns a **priority*** to each task based on the **level**, which is the greatest sum of execution times of tasks on a path in the precedence graph from the tasks to another task with no dependents;
- The scheduler assigns tasks to processors in the order of their priority as processors become available.

* – tasks with larger levels have higher priority than tasks with smaller levels.

#Consider previous example with precedence ($e_n = 1$):



1) Let's assign **levels**;
2) Then, from highest to lowest, schedule tasks as follows:

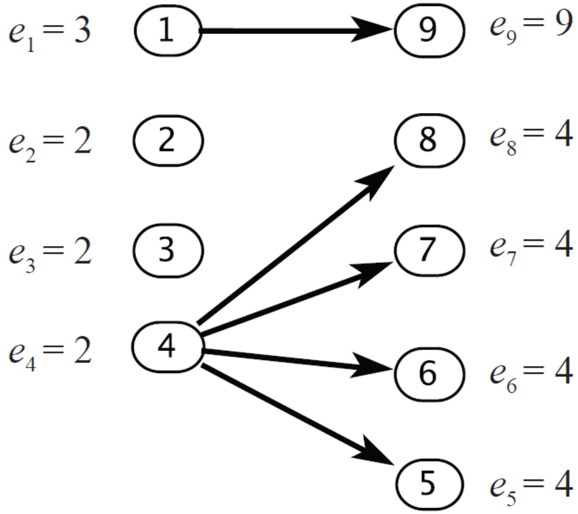


Scheduling anomalies

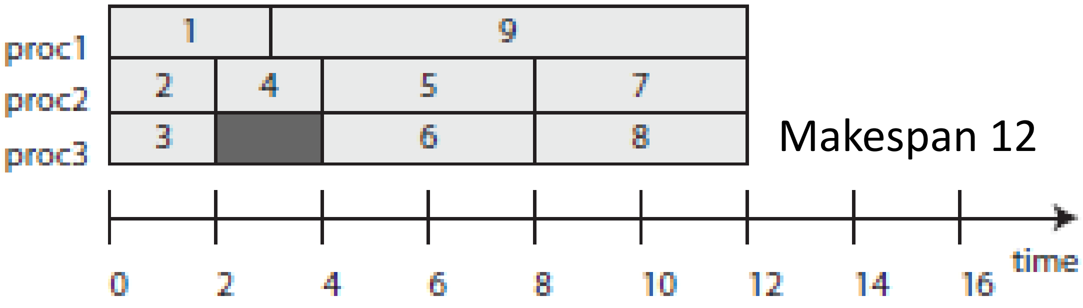
!When trying to improve performance, you may expect counterintuitive results from a scheduler, e.g., when trying:

- Increasing the number of processors;
- Reducing execution times;
- Weakening precedence constraints

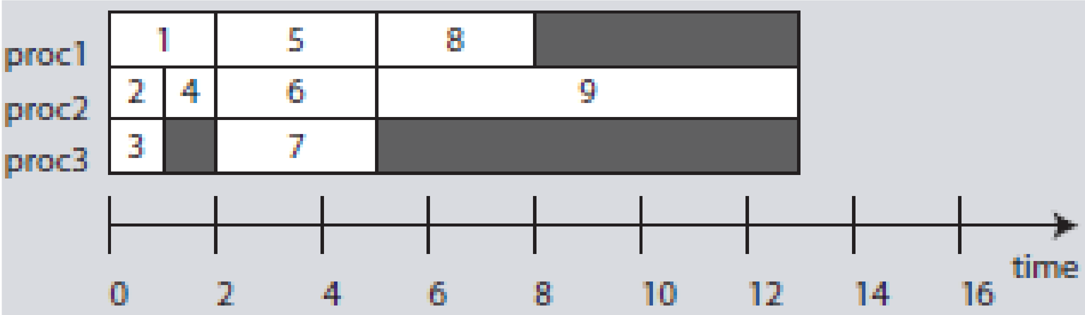
#Consider scenario, where tasks are assigned priorities according their number (lower number => higher priority):



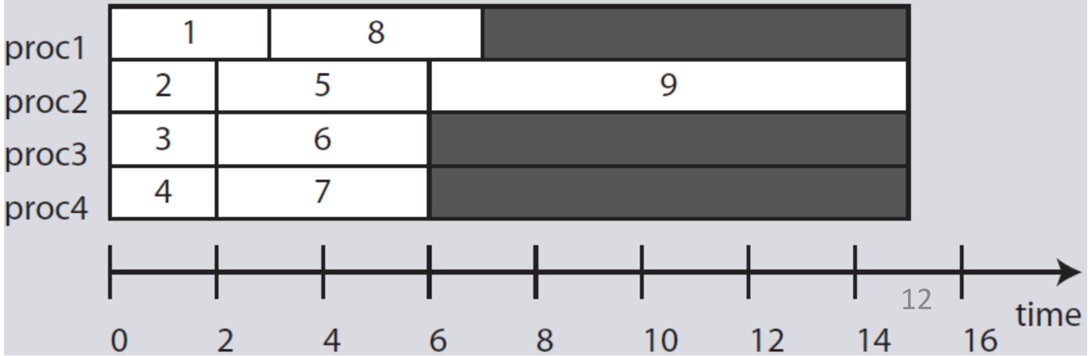
#Optimal priority-based non-preemptive schedule will look as follows (not **HLS**):



#If execution time of all tasks reduced by one (makespan 13):



#If we add fourth processor (makespan 15):



To sum up

- Scheduling is done either at program **design time** or **run time**;
- A **scheduling** includes three main **decisions**:
  - **Assignment** – which processor should execute the task;
  - **Ordering** – in what order each processor should execute its tasks;
  - **Timing** – the time at which each task executes.
- **Scheduler** uses a **task model**, i.e., the set of assumptions on task(s), to make an informed (hopefully, optimal) decisions;
- Main requirements for a **scheduler** are to be **simple**, **effective**, and **robust**;
- The choice of a **scheduling strategy** is governed by the goals of the application (e.g., **task model**) – there are effective scheduling strategies for a goal;
- When trying to improve performance, you may expect counterintuitive results from a scheduler – **scheduling anomalies**.

The end!

Assignment 2 deadline – **April 27, 23:59**:

- Only submit **.c** files with code in Embedded C (no reports).

See you next time – **May 4** (Tutorial 2).

Next lecture – **May 8**.