Introduction to Embedded Systems

6. Real-Time Scheduling

Prof. Dr. Marco Zimmerling

NETWORKED EMBEDDED SYSTEMS LAB

UNI FREIBURG

Organization

Lectures:

- Next week (December 20): virtually on Zoom (link will be posted on ILIAS)

Exercises:

- Due to timing constraints, we will no longer introduce the exercise sheets
- Today (December 13): no exercise
- Next week (December 20):
  - Solutions of fourth exercise sheet presented
  - Fifth exercise sheet released

Where we are …

Software

Hardware

1. Introduction to Embedded Systems
2. Software Development
3. Hardware-Software Interface
4. Programming Paradigms
5. Embedded Operating Systems
6. Real-time Scheduling
7. Shared Resources
8. Hardware Components
9. Power and Energy
10. Architecture Synthesis

Hardware-Software

Overview Aperiodic Task Scheduling

Scheduling of *aperiodic tasks* with real-time constraints:

- Table with some known algorithms:

		Equal arrival times non preemptive	Arbitrary arrival times preemptive
(*independent* = without precedence constraints)	**Independent tasks**	EDD (Jackson)	EDF (Horn)
(*dependent* = with precedence constraints)	**Dependent tasks**	LDF (Lawler)	EDF* (Chetto)

Overview

Table of some known *preemptive scheduling algorithms for periodic tasks*:

	$D_i = T_i$ Deadline equals period	$D_i \leq T_i$ Deadline smaller than period
static priority	RM (rate-monotonic)	DM (deadline-monotonic)
dynamic priority	EDF	EDF

Real-Time Scheduling of Mixed Task Sets

Mixed Task Sets: Motivation and Terminology

- Many applications feature both periodic and aperiodic tasks:
  - *Periodic tasks are time-driven* and execute critical activities (e.g., control) with hard timing constraints in order to guarantee regular activation rates of the tasks.
  - *Aperiodic tasks are usually event-driven* and may have hard, soft, or non-real-time requirements depending on the specific application.
- There are two types of hard aperiodic tasks:
  - *A sporadic task has a maximum arrival rate* or, equivalently, a minimum inter-arrival time between consecutive instances of the task. This is a worst-case assumption on the environment and allows for an offline guarantee on the task's schedulability.
  - *A firm task* is an aperiodic task for which the maximum arrival rate of the associated event cannot be bounded, but an online guarantee on the schedulability of individual instances of the task is still required. An online acceptance test is used to accept a request for a firm task only if the task instance can be served within its deadline.
- The objective is to meet the timing constraints of all critical periodic or aperiodic tasks and to provide small average response times for soft and non-real-time tasks.

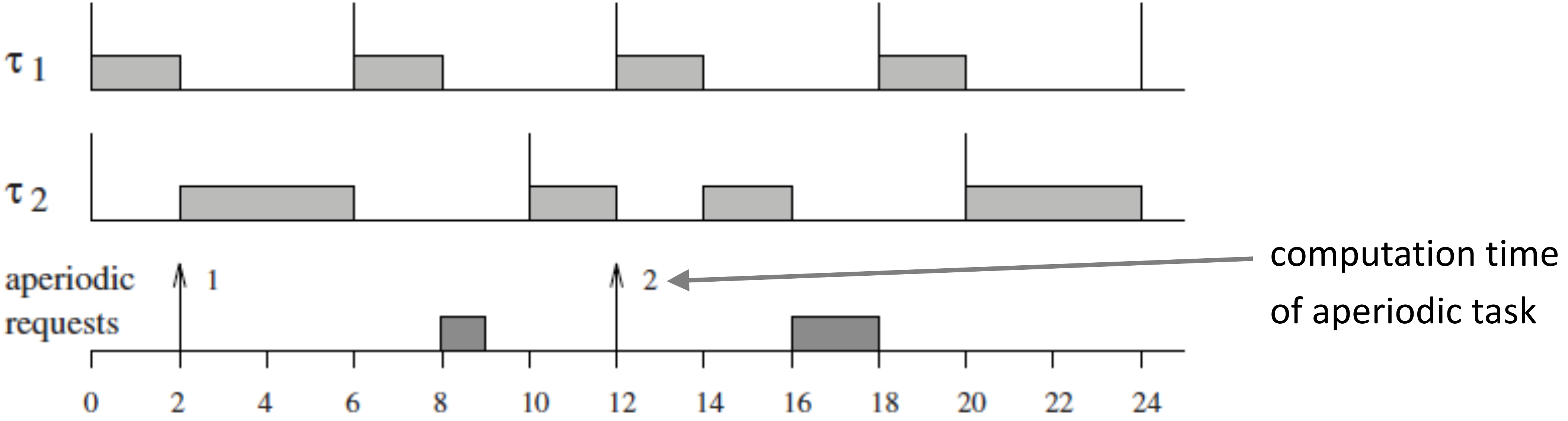
Assumptions

- Each periodic task has a relative deadline equal to its period, that is, $D_i = T_i$.
- All periodic tasks start simultaneously at time $t = 0$.
- The arrival times of aperiodic tasks are unknown.
- All tasks are fully preemptable.

Background Scheduling

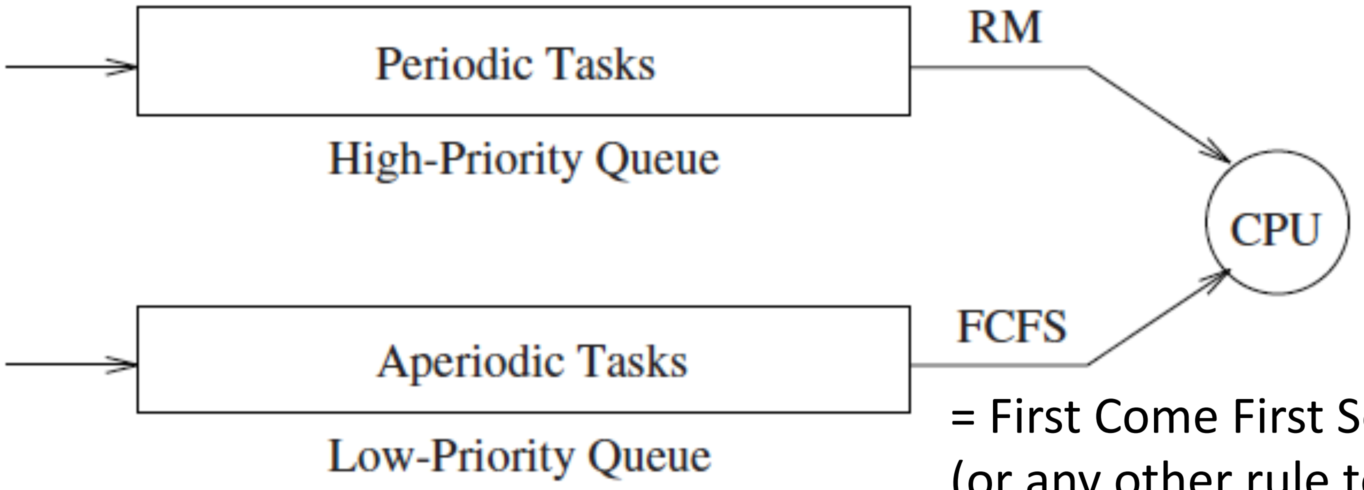
- *Principle*:
  - Periodic tasks are scheduled using RM
  - Aperiodic tasks are served whenever there are no periodic tasks ready to execute, that is, aperiodic tasks are ”scheduled in the background”

- *Example*: Two periodic tasks with $C_1 = 2, T_1 = 6$ and $C_2 = 4, T_2 = 10$



Background Scheduling

- *Advantages*:
  - Simplicity
  - Execution of periodic tasks is not affected



= First Come First Serve (or any other rule to decide on the order of execution)

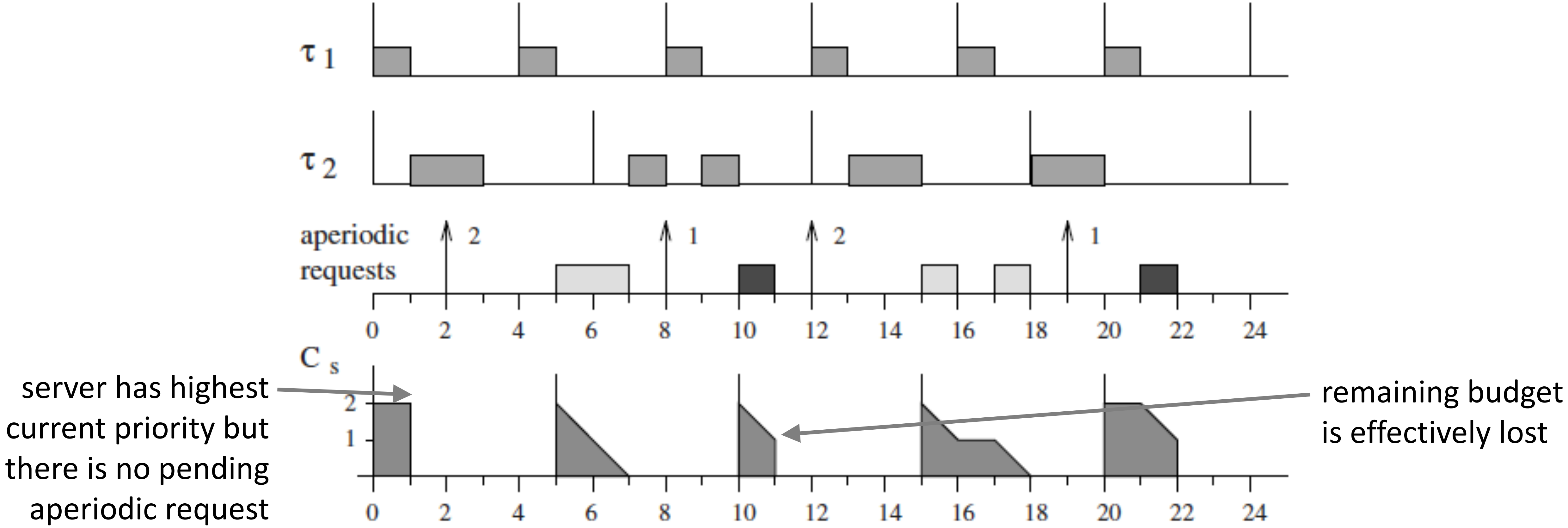
- *Disadvantages*:
  - The response time of aperiodic tasks can be prohibitively long. This problem arises in particular if the load induced by the periodic tasks is high.
  - There is no possibility to assign a higher priority to aperiodic tasks.

Polling Server

- *Idea*: Provide short average response times for aperiodic tasks by introducing an artificial periodic task that "serves" aperiodic requests as soon as possible
- *Principle*:
  - Introduce Polling Server (PS) task with period T_S and computation time C_S. The period T_S can be chosen to match the response time requirement of aperiodic tasks. The computation time C_S is called the *capacity* or the *budget* of the server.
  - Schedule PS task like any other periodic task, that is, using RM.
  - When PS has the highest current priority, it serves any pending aperiodic requests until its capacity is exhausted. 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.
- *Advantage*: Improves average response time compared to background scheduling
- *Disadvantage*: If an aperiodic requests arrives just after the server has suspended, it must wait until the beginning of the next polling period.

Polling Server: Example

- Two periodic tasks with $C_1 = 1, T_1 = 4$ and $C_2 = 4, T_2 = 6$
- PS task with $C_s = 2, T_s = 5$



Polling Server: Schedulability Analysis

- *Observation*: In the worst case, the server introduces the same interference as an equivalent periodic task with period T_s and computation time C_s.

- *Schedulability test*: If a set of n periodic tasks and the server are scheduled by RM, then the schedulability can be guaranteed if

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

utilization of
n periodic tasks

utilization of
the server

$n + 1$ because there are effectively
that many periodic tasks in the system

- Again, this schedulability test is *sufficient* but not necessary.

Polling Server: Schedulability of Firm Aperiodic Tasks

- At runtime, a request for a firm aperiodic task arrives with computation time C_a and relative deadline D_a. We need an acceptance test to check whether the request, which is handled by the server, can be completed within its deadline.

- If $C_a \le C_s$, schedulability is guaranteed if

$$2T_s \le D_a$$

- For arbitrary computation times, schedulability is guaranteed if

$$T_s + \left\lceil \frac{C_a}{C_s} \right\rceil T_s \le D_a$$

worst-case waiting time until the server becomes active

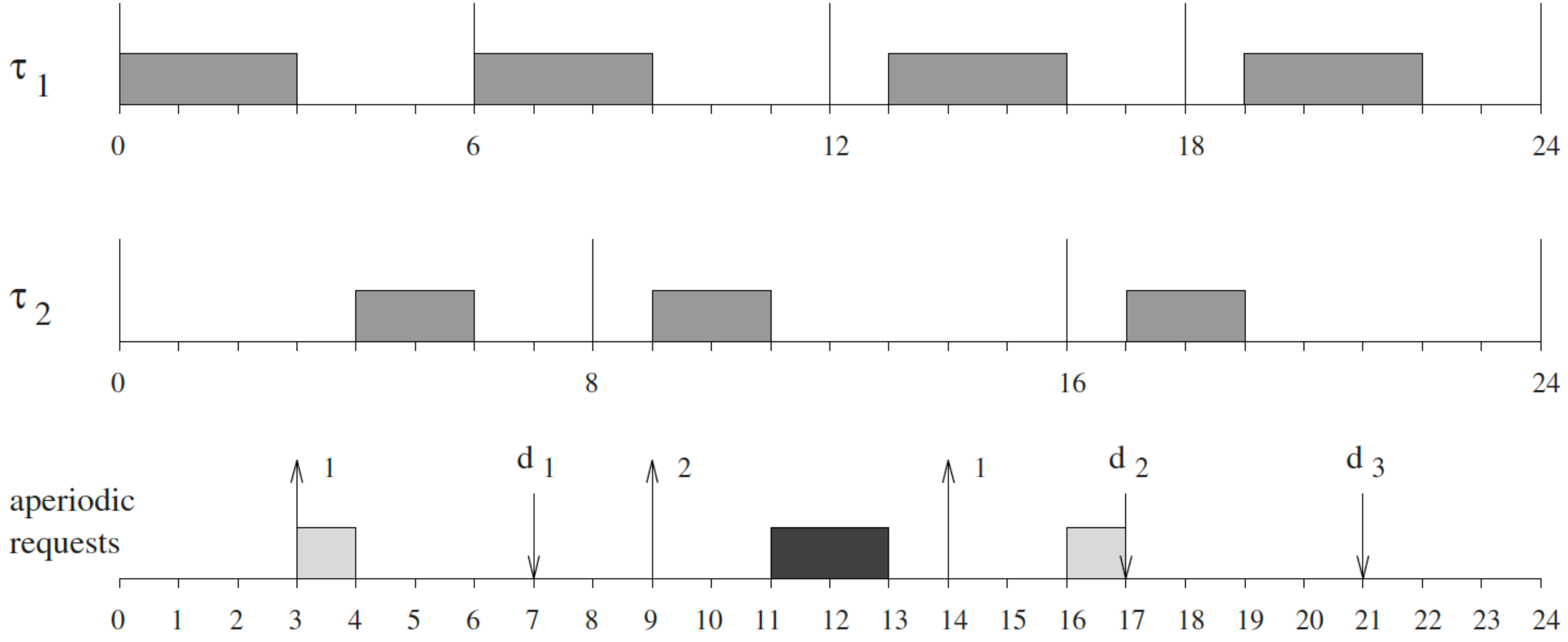
total number of server periods needed to process the request

Total Bandwidth Server

- *Idea*: Provide short average response times for aperiodic tasks by assigning a possible earlier deadline to aperiodic requests when they arrive
- *Principle*:
  - When the k-th aperiodic request arrives at time $t = r_k$, it receives a deadline
  $$d_k = \max(r_k, d_{k-1}) + \frac{C_k}{U_s}$$
  where C_k is the computation time of the request and U_s is the server utilization factor (that is, its bandwidth). By definition, $d_0 = 0$.
  - Once the deadline is assigned, the request is inserted into the ready queue of the system and scheduled by EDF like any other periodic task instance.
- *Advantages*:
  - Each time an aperiodic request arrives, total bandwidth of the server is immediately assigned to it, whenever possible. This can reduce the average response time.
  - Implementation overhead is practically negligible

Total Bandwidth Server: Example

- Two periodic tasks with $C_1 = 3, T_1 = 6$ and $C_2 = 2, T_2 = 8$
- Total Bandwidth Server (TBS) with $U_s = 1 - U_p = 0.25$



$$d_1 = r_1 + \frac{C_1}{U_s} = 7 \qquad d_2 = r_2 + \frac{C_2}{U_s} = 17 \qquad d_3 = \max(r_3, d_2) + \frac{C_3}{U_s} = 21$$

Total Bandwidth Server: Schedulability Analysis

- Given a set of n periodic tasks with implicit deadlines (i.e., $D_i = T_i$ for all tasks), processor utilization $\sum_{i=1}^{n} C_i/T_i = U_p$, and a TBS with processor utilization U_s, then the whole set is schedulable by EDF if and only if

$$U_p + U_s \leq 1$$

- This condition is both *necessary and sufficient*.