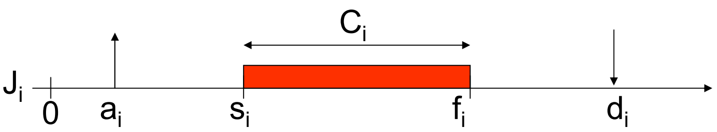
Embedded Systems 15

A-periodic scheduling REVIEW



- Given:
  - A set of non-periodic tasks $\{J_1, \ldots, J_n\}$ with
    - arrival times a_i, deadlines d_i, computation times C_i
    - precedence constraints
    - resource constraints
  - Class of scheduling algorithm:
    - Preemptive, non-preemptive
    - Off-line / on-line
    - Optimal / heuristic
    - One processor / multi-processor
    - …
  - Cost function:
    - Minimize maximum lateness (soft RT)
    - Minimize maximum number of late tasks (feasibility! – hard RT)
- Find:
  Optimal / good schedule according to given cost function

Case 1: Aperiodic tasks with synchronous release

REVIEW

- A set of (a-periodic) tasks $\{J_1, \ldots, J_n\}$ with
  - arrival times $a_i = 0 \ \forall \ 1 \leq i \leq n$, i.e. "synchronous" arrival times
  - deadlines d_i,
  - computation times C_i
  - no precedence constraints, no resource constraints, i.e. "independent tasks"
- non-preemptive
- single processor
- Optimal
- Find schedule which minimizes maximum lateness (variant: find feasible solution)

EDD – Earliest Due Date

REVIEW

EDD: execute the tasks in order of non-decreasing deadlines

- **Lemma**:
  If arrival times are synchronous, then preemption does not help, i.e. if there is a preemptive schedule with maximum lateness L_{max}, then there is also a non-preemptive schedule with maximum lateness L_{max}.
- **Theorem (Jackson '55):**
  Given a set of n independent tasks with synchronous arrival times, any algorithm that executes the tasks in order of non-decreasing deadlines is optimal with respect to minimizing the maximum lateness.

Case 2: aperiodic tasks with asynchronous release

REVIEW

- A set of (a-periodic) tasks $\{J_1, \ldots, J_n\}$ with
  - **arbitrary** arrival times a_i
  - deadlines d_i,
  - computation times C_i
  - no precedence constraints, no resource constraints, i.e. "independent tasks"
- **preemptive**
- Single processor
- Optimal
- Find schedule which minimizes maximum lateness (variant: find feasible solution)

EDF – Earliest Deadline First

REVIEW

- EDF: At every instant execute the task with the earliest absolute deadline among all the ready tasks.
- **Theorem (Horn '74):**
  Given a set of n independent task with arbitrary arrival times, any algorithm that at every instant executes the task with the earliest absolute deadline among all the ready tasks is optimal with respect to minimizing the maximum lateness.

Non-preemptive version

REVIEW

- Changed problem:
  - A set of (a-periodic) tasks $\{J_1, \ldots, J_n\}$ with
    - **arbitrary** arrival times a_i
    - deadlines d_i,
    - computation times C_i
    - no precedence constraints, no resource constraints, i.e. “independent tasks”
  - **Non-preemptive** instead of preemptive scheduling!
  - Single processor
  - Optimal
  - Find schedule which minimizes maximum lateness (variant: find feasible solution)

Non-preemptive version

REVIEW

- **Theorem** (Jeffay et al. ’91): EDF is an optimal *non-idle* scheduling algorithm also in a non-preemptive task model.

- When idle schedules are allowed: problem is NP-hard.
- Possible approaches:
  - Heuristics
  - Bratley’s algorithm: branch-and-bound

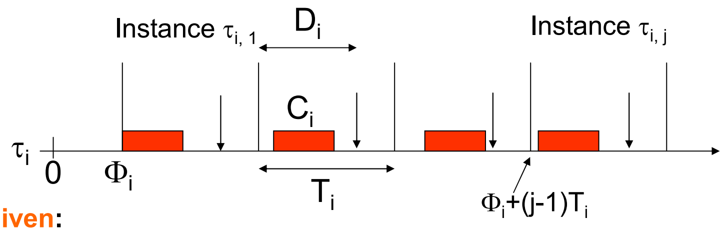
Case 3: Scheduling with precedence constraints

REVIEW

- Non-preemptive scheduling with non-synchronous arrival times, deadlines and precedence constraints is NP-hard.
- Restrictions:
  - Consider synchronous arrival times (all tasks arrive at 0)
  - Allow preemption.
- **Theorem (Lawler 73):**
  LDF (Latest Deadline First) is optimal wrt. maximum lateness.

Optimal scheduling algorithms for *periodic* tasks

Periodic scheduling



- **Given:**
  - A set of periodic tasks $\Gamma = \{\tau_1, \ldots, \tau_n\}$ with
    - phases Φ_i (arrival times of first instances of tasks),
    - periods T_i (time difference between two consecutive activations)
    - relative deadlines D_i (deadline relative to arrival times of instances)
    - computation times C_i

  $\Rightarrow$ *j* th instance $\tau_{i,j}$ of task τ_i with
    - arrival time $a_{i,j} = \Phi_i + (j-1)\,T_i$,
    - deadline $d_{i,j} = \Phi_i + (j-1)\,T_i + D_i$,
    - start time $s_{i,j}$ and
    - finishing time $f_{i,j}$
- **Find a feasible schedule**

Assumptions

A.1. Instances of periodic task τ_i are regularly activated with constant period T_i.

A.2. All instances have same worst case execution time C_i.

A.3. All instances have same relative deadline D_i, here in most cases equal to T_i (i.e., $d_{i,j} = \Phi_i + j \cdot T_i$)

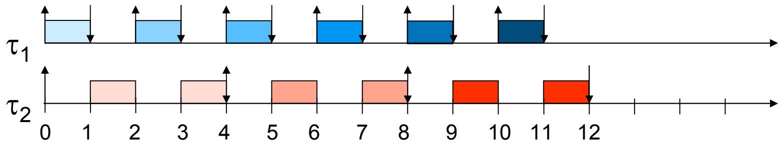
A.4. All tasks in Γ are independent. No precedence relation, no resource constraints.

A.5. Overhead for context switches is neglected, i.e. assumed to be 0 in the theory.

- Basic results based on these assumptions form the core of scheduling theory.
- For practical applications, assumptions A.3. and A.4. can be relaxed, but results have to be extended.

Examples for periodic scheduling (1)

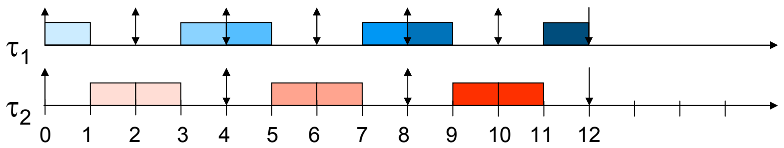
	τ_1	τ_2
Φ_i	0	0
T_i	2	4
C_i	1	2
D_i	1	4



- Schedulable, but only preemptive schedule possible.

Examples for periodic scheduling (2)

	τ_1	τ_2
Φ_i	0	0
T_i	2	4
C_i	1	2
D_i	2	4



- Schedulable with non-preemptive schedule.

Examples for periodic scheduling (3)

	τ_1	τ_2
Φ_i	0	0
T_i	3	4
C_i	2	2
D_i	3	4

T_1: 12/3 = 4

T_2: 12/4 = 3

- No feasible schedule for single processor.

During $T_1 \cdot T_2 = 3 \cdot 4 = 12$

- 4 activations of T_1 : 8 units
- 3 activations of T_2 : 6 units

⇒ 14 units of CPU time in 12 time units ↯ 14 units

Processor utilization

Definition:

Given a set Γ of n periodic tasks, the **processor utilization U** is given by

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i}.$$

Processor utilization: using it as a schedulability criterion

- Given: a scheduling algorithm A
- Define $U_{bnd}(A) = \inf \{U(\Gamma) \mid \Gamma$ is not schedulable by algorithm A$\}$.
- If $U_{bnd}(A) > 0$ then a simple, sufficient criterion for schedulability by A can be based on processor utilization:
  - If $U(\Gamma) < U_{bnd}(A)$ then Γ is schedulable by A.
  - However, if $U_{bnd}(A) < U(\Gamma) \leq 1$, then Γ may or may not be schedulable by A.
- **Question**:
  Does a scheduling algorithm A exist with $U_{bnd}(A) = 1$?

Processor utilization

- **Question**:
  Does a scheduling algorithm A exist with $U_{bnd}(A) = 1$?
- **Answer:**
  - No, if $D_i < T_i$ allowed.
  - Example:

	τ_1	τ_2
Φ_i	0	0
T_i	2	2
C_i	1	1
D_i	1	1

  - Yes, if $D_i = T_i$ (or $D_i \geq T_i$) $\Rightarrow$ Earliest Deadline First (EDF)
  - In the following: assume $D_i = T_i$

Earliest Deadline First (EDF)

- EDF is applicable to both periodic and a-periodic tasks.
- If there are only periodic tasks, priority-based schemes like "rate monotonic scheduling (RM)" (see later) are often preferred, since
  - They are simpler due to fixed priorities
    $\Rightarrow$ use in "standard OS" possible
  - sorting wrt. to deadlines at run time is not needed

EDF and processor utilization factor

- **Theorem: A set of periodic tasks $\tau_1, \ldots, \tau_n$ with $D_i = T_i$ is schedulable with EDF iff $U = \sum_{i=1}^{n} C_i / T_i \leq 1$.**

"$\Rightarrow$": Let $T = T_1 \cdot \ldots \cdot T_n$

$U > 1 \Rightarrow UT > T$

$\Rightarrow \sum_{i=1}^{n} \frac{C_i}{T_i} T > T$

$\Rightarrow \sum_{i=1}^{n} \frac{T}{T_i} \cdot C_i > T$

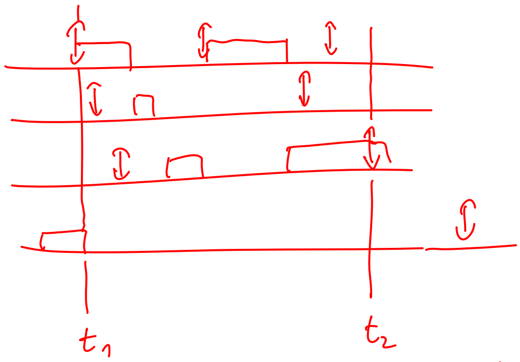
$\frac{T}{T_i}$: # of times T_i is scheduled in T

$\frac{T}{T_i} \cdot C_i$: total time spent on T_i in T

$\sum_{i=1}^{n} \frac{T}{T_i} \cdot C_i$: Total processing time

"⇐"

- Assume $U \leq 1$ and task set is not EDF schedulable.
- Let t_2 be the earliest time in EDF-schedule where a task misses the deadline



- Let $[t_1, t_2]$ be the largest interval s.t.
  - $[t_1, t_2]$ does not contain idle times
  - only instances with deadlines $\leq t_2$ are executed.

<u>Claim:</u> The tasks executed in $[t_1, t_2]$ have arrival times $\geq t_1$ and deadlines $\leq t_2$

- $\leq t_2$ by construction
- $\geq t_1$:

Case 1: The processor was idle directly before t_1
$\Rightarrow$ In the EDF schedule there is no unfinished task with arrival $< t_1$

Case 2: The task running directly before t_1 has deadline $\leq t_2$
$\rightarrow$ Contradiction to maximality of $[t_1, t_2]$

Case 3: The task running directly before t_1 has deadline $> t_2$
$\rightarrow$ Due to EDF there is no task with arrival $< t_1$ and deadline $\leq t_2$

Since all tasks in $[t_1, t_2]$ have deadline $\leq t_2$ $\rightarrow$ all tasks have been arrived $\geq t_1$. □ Claim

There is a time overflow at t_2, there is no idle time between t_1 and t_2

$$\Rightarrow (t_2 - t_1) < \sum_{a_{i,j} \geq t_1,\, d_{i,j} \leq t_2} C_i$$

$$= \sum_{i=1}^{n} \left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor C_i$$

$$\leq \sum_{i=1}^{n} \frac{t_2 - t_1}{T_i} C_i$$

$$= (t_2 - t_1) \sum_{i=1}^{n} \frac{C_i}{T_i}$$

$$= (t_2 - t_1) \cdot U$$

$\Rightarrow U > 1.$ ↯.

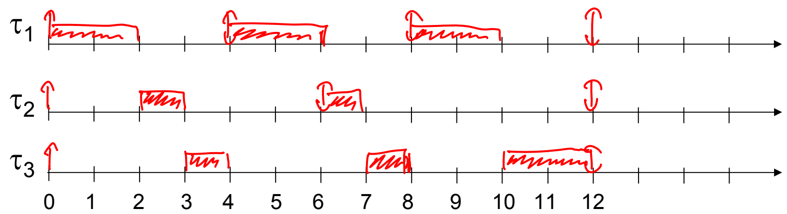
Rate monotonic scheduling (RM)

- Rate monotonic scheduling (RM) (Liu, Layland ’73):
  - Assign fixed priorities to tasks τ_i:
    - priority(τ_i) = $1/T_i$
    - I.e., priority reflects release rate
  - Always execute ready task with highest priority
  - Preemptive: currently executing task is preempted by newly arrived task with shorter period.

Example for RM (1)

	τ_1	τ_2	τ_3
Φ_i	0	0	0
T_i	4	6	12
C_i	2	1	4
D_i	4	6	12

priority(τ_1)
> priority(τ_2)
> priority(τ_3)

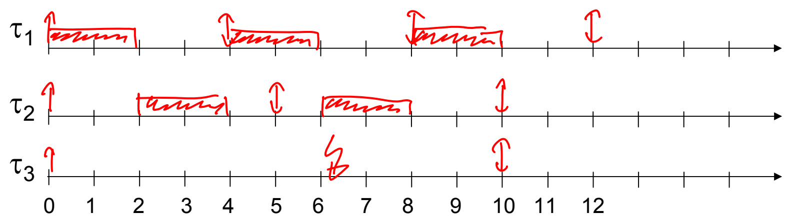


Example for RM (2)

	τ_1	τ_2	τ_3
Φ_i	0	0	0
T_i	4	5	10
C_i	2	2	1
D_i	4	5	10

$U = \frac{2}{4} + \frac{2}{5} + \frac{1}{10}$

$= 1$



No feasible schedule!

Optimality of Rate Monotonic Scheduling

- **Theorem (Liu, Layland, 1973):**
  RM is optimal among all fixed-priority scheduling algorithms.

- **Def.:** The **response time $R_{i,j}$** of an instance j of task i is the time (measured from the arrival time) at which the instance is finished: $R_{i,j} = f_{i,j} - a_{i,j}$.
- The critical instant of a task is the time at which the arrival of the task will produce the largest response time.

Response times and critical instants

- **Observation**:
  For RM, the critical instant t of a task τ_i is given by the time when $\tau_{i,j}$ arrives together with all tasks $\tau_1, \ldots, \tau_{i-1}$ with higher priority.

· Let C_k be the computation time of task τ_k

· Let $c_k(t)$ be the remaining time for the last instance of τ_k which arrived before t

· Response time for $\tau_{i,j}$

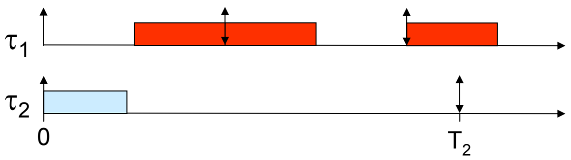
$$\sum_{k=1}^{i-1} \left(c_k(t) + (\# \text{ of arrivals of } \tau_k \text{ before finishing time of } \tau_{i,j}) \cdot C_k \right) + C_i$$

=> Response time is maximal if $c_k(t) = C_k$.

Response times and critical instants

- For our "worst case task sets" we can assume that there are critical instants where an instance of a task arrives together with all higher priority tasks.
- A task set is schedulable, if the response time at these critical instants is not larger than the relative deadline.

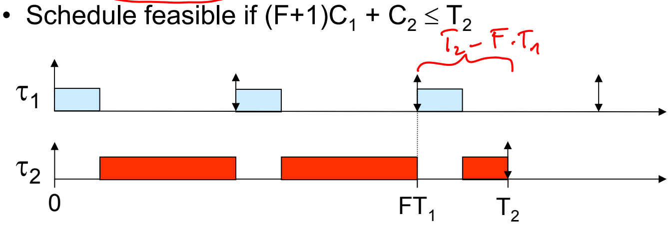
Non-RM Schedule



Schedule feasible iff $C_1 + C_2 \leq T_1$

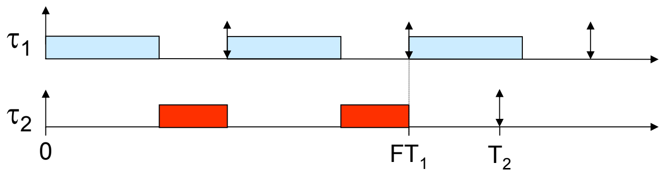
RM-Schedule

- Let $F = \lfloor T_2 / T_1 \rfloor$ be the number of periods of τ_1 entirely contained in T_2.
- Case 1:
  - The computation time C_1 is short enough, so that all requests of τ_1 within period of τ_2 are completed before second request of τ_2.
  - I.e. $C_1 \leq T_2 - F\,T_1$
  - Schedule feasible if $(F+1)C_1 + C_2 \leq T_2$



RM-Schedule

- Case 2:
  - The second request of τ_2 arrives when τ_1 is running.
  - I.e. $C_1 \geq T_2 - F\,T_1$



Schedule feasible if $FC_1 + C_2 \leq FT_1$

Proof of Liu/Layland (for 2 tasks)

We show that if task set is schedulable by non-RM then it is schedulable by RM, i.e.

Case 1: $C_1 \le T_2 - FT_1$:

$C_1 + C_2 \le T_1 \overset{?}{\Rightarrow} (F+1)C_1 + C_2 \le T_2$

$\Downarrow$

$FC_1 + FC_2 \le FT_1$

$\Downarrow$ $F \ge 1$

$FC_1 + C_2 \le FT_1$

$\Downarrow$ $+C_1$

$(F+1)C_1 + C_2 \le FT_1 + C_1$

$\Downarrow$ $C_1 \le T_2 - FT_1$

$(F+1)C_1 + C_2 \le T_2$

Case 2: $C_1 \ge T_2 - FT_1$

We show $C_1 + C_2 \le T_1 \Rightarrow FC_1 + C_2 \le FT_1$

$\Downarrow$

$FC_1 + FC_2 \le FT_1$

$\Downarrow$ $F \ge 1$

$FC_1 + C_2 \le FT_1$