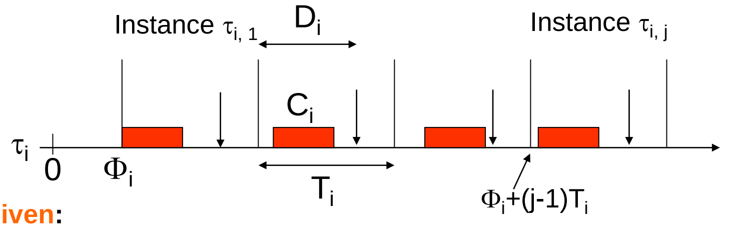
Embedded Systems 16

Info

- Midterm exam inspection
  tomorrow Friday June 22 2-3pm
  in room 528, E1.3
- Lecture on Tuesday June 26
  in HS 1 / Mathematics

REVIEW: 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 $d_{i,j} = \Phi_i + (j-1)\, T_i + D_i$,
- **Find a feasible schedule**
    - start time $s_{i,j}$ and
    - finishing time $f_{i,j}$

REVIEW: Processor utilization

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

- 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.
- **Theorem:** A set of periodic tasks $\tau_1, \ldots, \tau_n$ with $D_i = T_i$ is schedulable with EDF iff $U \leq 1$.

REVIEW: 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.

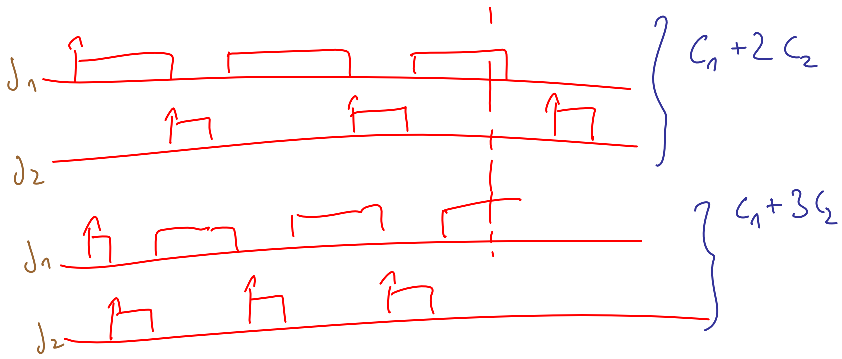
Optimality of Rate Monotonic Scheduling

- **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.

REVIEW: Response times and critical instants

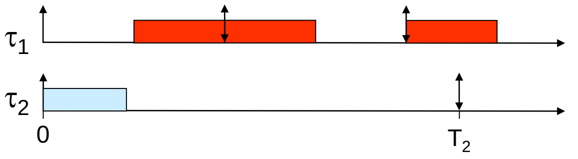
- **Observation**:
  For RM, the critical instant t of a task τ_i is given by the time when $\tau_{i,\,j}$ arrives together with all tasks $\tau_1, \ldots, \tau_{i-1}$ with higher priority.



Response times and critical instants

- For our “worst case task sets” we focus on the 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.

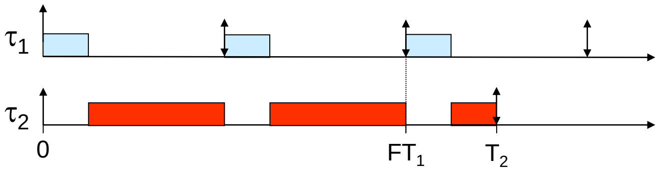
Non-RM Schedule



Schedule feasible iff $C_1 + C_2 \leq T_1$

RM-Schedule

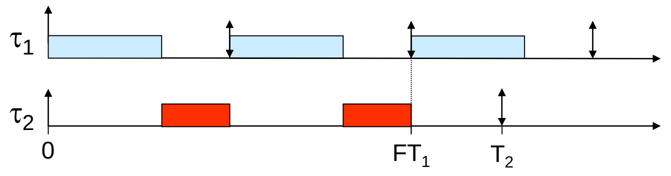
- Let $F = \lfloor T_2 / T_1 \rfloor$ be the number of periods of τ_1 entirely contained in T_2.
- **Case 1:**
  - The computation time C_1 is short enough, so that all requests of τ_1 within period of τ_2 are completed before second request of τ_2.
  - I.e. $C_1 \leq T_2 - F\,T_1$



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

We show: If task set is schedulable by Non-RM
⇒ schedulable by RM

Case 1: $C_1 \le T_2 - F T_1$

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

$\Rightarrow F \cdot C_1 + F C_2 \le F T_1$

$\Rightarrow F \cdot C_1 + C_2 \le F T_1$ ($F \geq 1$)

$\Rightarrow (F+1)\, C_1 + C_2 \le F T_1 + C_1$

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

$\Rightarrow C_1 \le T_2 - F T_1$

Case 2: $C_1 \geq T_2 - F T_1$

$C_1 + C_2 \leq T_1 \quad \overset{?}{\Rightarrow} \quad F C_1 + C_2 \leq F T_1$

$\Rightarrow F C_1 + F C_2 \leq F T_1$

$\Rightarrow F C_1 + C_2 \leq F T_1$

$F \geq 1$

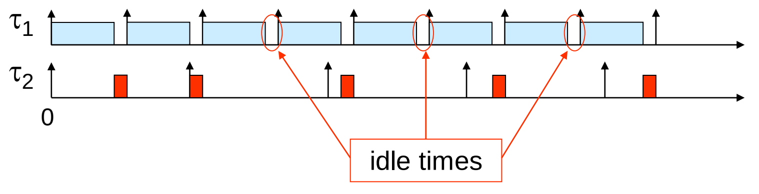
Computation of U_{bnd}(RM)

- We focus on task sets with 2 tasks (general case: n tasks)
- Computation of
  $U_{bnd}(RM, 2) = \inf \{U(\Gamma) \mid \Gamma \text{ is not schedulable by RM}, |\Gamma| = 2\}$.

- **Idea:**
  - Construct set of tasks with following properties:
    1. Set of tasks is schedulable by RM.
    2. Any increase of computation times makes the set of tasks non-schedulable.
    3. Processor utilization is minimal under properties 1. and 2.

Computation of U_{bnd}(RM, 2)

Worst case situation constructed for 2 processes:

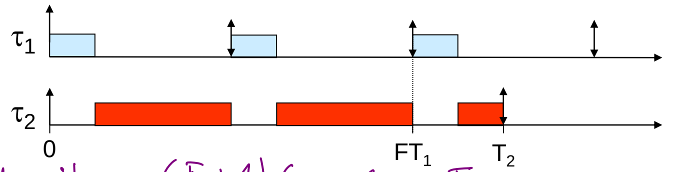


Computation of U_{bnd}(RM, 2)

- Consider a set of 2 periodic tasks τ_1 and τ_2 with $T_1 \leq T_2$ $\Rightarrow$ priority(τ_1) > priority(τ_2).
- We consider the critical instant when τ_1 and τ_2 arrive at the same time.
- We construct a worst case scenario where any increase of computation times destroys schedulability and minimize the processor utilization.

  This is done by manipulating
  - computation times C_1 and C_2 and
  - T_1 and T_2 (more precisely T_2 / T_1)

Case 1: $C_1 \le T_2 - F\,T_1$



feasible if $(F+1)\,C_1 + C_2 \le T_2$

max value of C_2: $C_2 = T_2 - (F+1)\,C_1$

Minimize U by adjusting C_1:

$$U = \frac{C_1}{T_1} + \frac{C_2}{T_2} = \frac{C_1}{T_1} + \frac{T_2 - (F+1)\,C_1}{T_2} =$$

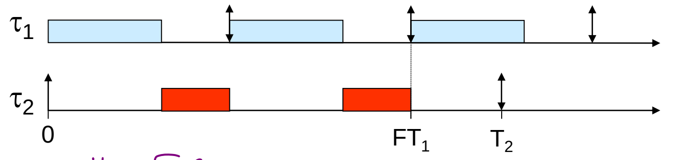
$$= 1 + \frac{C_1}{T_1} - \frac{C_1}{T_2}(F+1)$$

$$= 1 + \frac{C_1}{T_2}\underbrace{\left(\frac{T_2}{T_1} - \left(\left\lfloor \frac{T_2}{T_1} \right\rfloor + 1\right)\right)}_{<0}$$

$\Rightarrow$ U decreases monotonically with C_1

$\Rightarrow$ U minimized for $C_1 = T_2 - FT_1$ (in case 1)

Case 2: $C_1 \geq T_2 - F\,T_1$



feasible if $F C_1 + C_2 \leq F T_1$

$\Rightarrow$ Max value for C_2: $C_2 = (T_1 - C_1) \cdot F$

$$U = \frac{C_1}{T_1} + \frac{C_2}{T_2} = \frac{C_1}{T_1} + \frac{(T_1 - C_1)\cdot F}{T_2} =$$

$$= \frac{T_1}{T_2} F + \frac{C_1}{T_2} \underbrace{\left(\frac{T_2}{T_1} - F \right)}_{>0} \qquad (*)$$

$\Rightarrow$ U increases monotonically with C_1

$\Rightarrow$ U is minimal for $C_1 = T_2 - F T_1$ (in case 2)

$\Rightarrow$ $\underline{C_1 = T_2 - F T_1}$ in both cases

Manipulating T_2/T_1

$$U = \frac{T_1}{T_2} F + \frac{c_1}{T_2}\left(\frac{T_2}{T_1} - F\right) \qquad (*)$$

$$= \frac{T_1}{T_2} F + \frac{T_2 - F T_1}{T_2}\left(\frac{T_2}{T_1} - F\right)$$

$c_1 = T_2 - F T_1$

$$= \frac{T_1}{T_2}\left[F + \left(\frac{T_2}{T_1} - F\right)\left(\frac{T_2}{T_1} - F\right)\right]$$

(Let $G = \frac{T_2}{T_1} - F$)

$$= \frac{T_1}{T_2}\left(F + G^2\right)$$

$$U = \frac{T_1}{T_2}\left(F + G^2\right) =$$

$$= \frac{F + G^2}{\frac{T_2}{T_1}} = \frac{F + G^2}{F + G} \qquad (**)$$

$$= \frac{F + G - (G - G^2)}{F + G} = 1 - \frac{G(1-G)}{F+G}$$

Since $G = \frac{T_2}{T_1} - \left\lfloor \frac{T_2}{T_1} \right\rfloor$, $0 \le G < 1$

$\Rightarrow G(1-G) \geq 0$

$\Rightarrow$ U increases monotonically with F

$\Rightarrow$ U minimal for minimal value of F

$F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$, $T_1 \le T_2$ $\Big\}$ $\Rightarrow F = 1$

$$U = \frac{F + G^2}{F + G} \quad (**), \quad F = 1$$

$$= \frac{1 + G^2}{1 + G}$$

Minimize U over G:

$$\frac{dU}{dG} = \frac{2G \cdot (1+G) - (1+G^2)}{(1+G)^2} = \frac{G^2 + 2G - 1}{(1+G)^2}$$

$$\frac{dU}{dG} = 0 \Rightarrow G^2 + 2G - 1 = 0$$

$$G \in \left\{ \frac{-2 + \sqrt{4+4}}{2}, \frac{-2 - \sqrt{4+4}}{2} \right\}$$

$$G \in \{ -1 + \sqrt{2}, -1 - \sqrt{2} \}$$

$$\Rightarrow G = -1 + \sqrt{2}, \text{ since } 0 \leq G.$$

$$U = \frac{1 + G^2}{1 + G} = \frac{1 + (-1 + \sqrt{2})^2}{1 + (-1 + \sqrt{2})} =$$

$$= \frac{4 - 2\sqrt{2}}{\sqrt{2}} = 2(\sqrt{2} - 1) \approx \underline{\underline{0.83}}$$

Computation of U_{bnd}(RM)

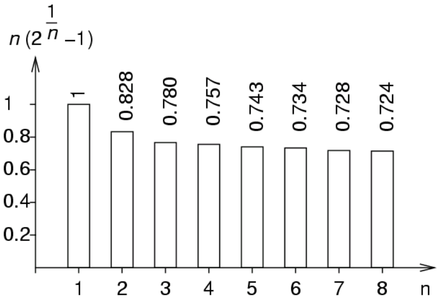
- Result for two processes:
  Any set of two periodic tasks with a processor utilization factor ≤ $\boxed{U_{bnd} = 2(2^{1/2} - 1)}$ can be scheduled by RM.

- Similarly, for the general case of **n** processes the following can be shown:
  Any set of n periodic tasks with a processor utilization factor ≤ $\boxed{U_{bnd} = n(2^{1/n} - 1)}$ can be scheduled by RM.

Computation of U_{bnd}(RM)

- Any set of n periodic tasks with a processor utilization factor ≤ $U_{bnd} = n(2^{1/n} - 1)$ can be scheduled by RM.

- U_{bnd} is decreasing with n and converges to ln 2 ≈ 0.69 for n → ∞



Schedulability check

- Hence, a set of tasks can be scheduled by RM if
  $U < U_{bnd}(RM) = \ln 2 \approx 0.69$

- But what can we say about schedulability when processor utilization factor is larger than $n(2^{1/n}-1)$?

- We can compute a more precise result, if we make use of the knowledge of periods T_i and computation times C_i.

Schedulability check

- Compute an upper bound R_i on the response time:
  - Suppose that $\tau_1, ..., \tau_n$ are ordered with increasing periods (i.e. decreasing priorities).
  - Consider an arbitrary periodic task τ_i.
  - At a critical instant t, when an instance of τ_i arrives together with all higher priority tasks, we have:
    - $R_i = C_i + \sum_{k=1}^{i-1}$ (# activations of τ_k during $[t, t + R_i]$) $\cdot C_k$
      $= C_i + \sum_{k=1}^{i-1} \lceil R_i/T_k \rceil \cdot C_k$

Schedulability check

- Compute the following sequence:
  - $R_i^{(0)} = C_i$.
  - $R_i^{(j+1)} = C_i + \sum_{k=1}^{i-1} \lceil R_i^{(j)} / T_k \rceil \cdot C_k$.
- It is easy to see that this sequence is monotonically increasing, i.e., $f(x) = C_i + \sum_{k=1}^{i-1} \lceil x / T_k \rceil \cdot C_k$ is monotonically increasing.
- $\Rightarrow$ If a least fixed point of f(x) exists, then the sequence converges to this fixed point.

Schedulability check

Algorithm:

$\forall$ i: $R_i^{(0)} = C_i$
repeat
  $\forall$ i: $R_i^{(j+1)} = C_i + \sum_{k=1}^{i-1} \lceil R_i^{(j)} / T_k \rceil \cdot C_k$
until ($\exists$ i with $R_i^{(j+1)} > D_i$) **or** ($\forall$ i $R_i^{(j+1)} = R_i^{(j)}$);
if ($\forall$ i $R_i^{(j+1)} = R_i^{(j)}$) **then**
  report (“RM schedulable”);

Example

	τ_1	τ_2	τ_3	τ_4
T_i	4	5	6	11
C_i	1	1	2	1
D_i	3	4	5	10

$R_1^0 = 1$, $R_2^0 = 1$, $R_3^0 = 2$, $R_4^0 = 1$

$R_2^1 = 2$, $R_3^1 = 4$, $R_4^1 = 5$

$R_2^2 = 2$, $R_3^2 = 4$, $R_4^2 = 6$

$R_4^3 = 7$

$R_4^4 = 9$

$R_4^5 = 10$

$R_4^6 = 10$

$\Rightarrow$ RM-schedulable.

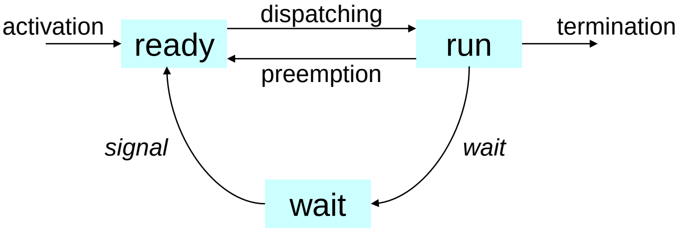
Summary

- Problem of scheduling independent and preemptable periodic tasks
- **Rate monotonic scheduling:**
  - Optimal solution among all fixed-priority schedulers
  - Schedulability of n tasks guaranteed, if processor utilization $U \leq n(2^{1/n} - 1)$.
- **Earliest deadline first:**
  - Optimal solution among all dynamic-priority schedulers
  - Schedulability guaranteed if processor utilization $U \leq 1$.

Rate Monotonic Scheduling in Presence of Task Dependencies

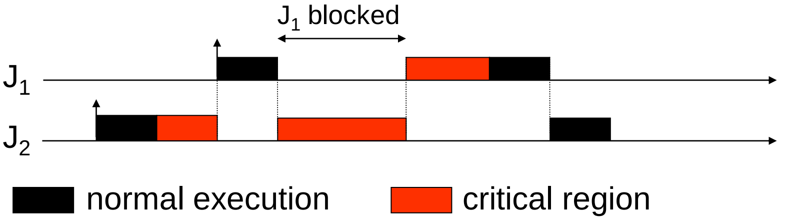
Wait state caused by resource constraints

- Each mutually exclusive **resource** R_i is protected by a **semaphore** S_i.
- Each critical section operating on R_i must begin with a *wait*(S_i) primitive and end with a *signal*(S_i) primitive.
- *wait* primitive on locked semaphore → wait state until another task executes signal primitive



The priority inversion problem

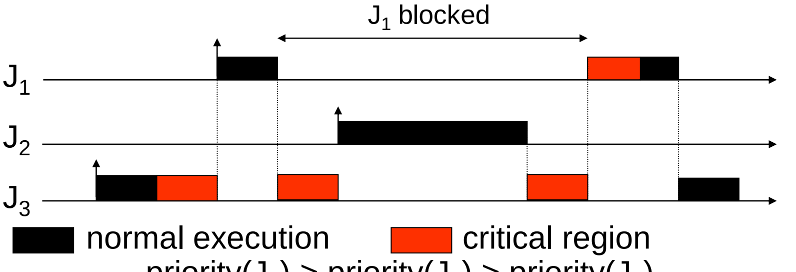
- **Priority inversion** can occur due to resource conflicts (exclusive use of shared resources) in fixed priority schedulers like RM:



priority(J_1) > priority(J_2)

- Here: Blocking time equal to length of critical section.

The priority inversion problem

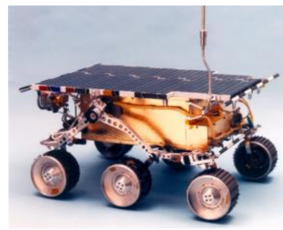


priority(J_1) > priority(J_2) > priority(J_3)

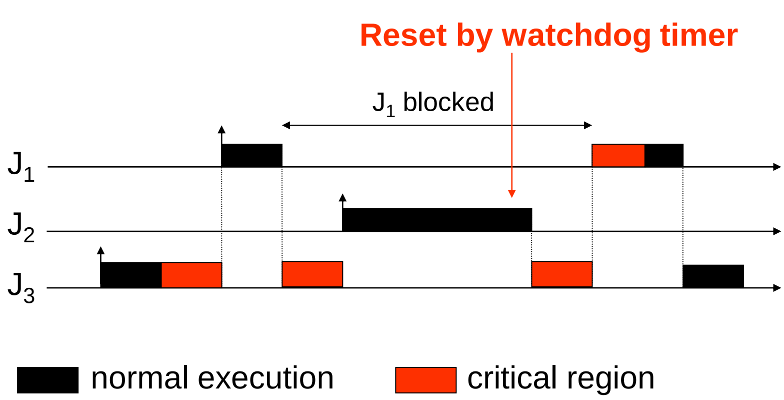
- Blocking time equal to length of critical section + computation time of J_2.
- Unbounded time of priority inversion, if J_3 is interrupted by tasks with priority between J_1 and J_3 during its critical region.

Priority inversion in real life: The MARS Pathfinder problem (1)

“But a few days into the mission, not long after Pathfinder started gathering meteorological data, the spacecraft began experiencing total system resets, each resulting in losses of data. The press reported these failures in terms such as "software glitches" and "the computer was trying to do too many things at once".” …



Priority inversion in real life: The MARS Pathfinder problem



priority(J_1) > priority(J_2) > priority(J_3)

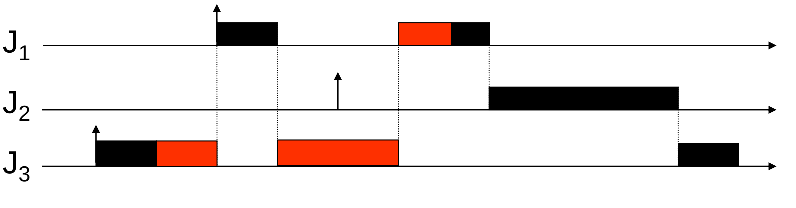
Coping with priority inversion: The priority inheritance protocol

Idea of **priority inheritance protocol**:

- If a task J_h blocks, since another task J_l with lower priority owns the requested resource, then J_l inherits the priority of J_h.
- When J_l releases the resource, the priority inheritance from J_h is undone.
- **Rule:** Tasks always inherit the highest priority of tasks blocked by it.

Direct vs. push-through blocking

- **Direct blocking:** High-priority job tries to acquire resource already held by lower-priority job
- **Push-through blocking:** Medium-priority job is blocked by lower-priority job that has inherited a higher priority.



Transitive priority inheritance