Real-Time Scheduling

Prof. Giorgio Buttazzo

Scuola Superiore Sant'Anna
Pisa, Italy

Email: giorgio.buttazzo@sssup.it

artist — scuola superiore Sant'Anna di studi universitari e di perfezionamento

Graduate Course on Embedded Real-Time Control Systems: Theory and Practice

Retis Real-Time Systems Laboratory

Scuola Superiore Sant'Anna, Pisa
14-18 June 2010

Monday, June 14

08:30 Overview of real-time scheduling (G. Buttazzo)
10:30 Coffee Break
11:00 Overview of the embedded platform (P. Gai)
13:00 Lunch Break
14:00 The ERIKA real-time kernel (P. Gai)
16:00 Break
16:30 Programming examples
18:30 End of Session

Tuesday, June 15

08:30 dsPic architecture and the Flex board (M. Marinoni)
10:30 Coffee Break
11:00 Lab practice with Flex and Erika
13:00 Lunch Break
14:00 Wireless communication (G. Franchino)
16:00 Break
16:30 Lab practice with wireless communication
18:30 End of Session

Wednesday, June 16

08:30 Introduction to real-time control (P. Marti)
10:30 Coffee Break
11:00 Feedback scheduling (M. Velasco)
13:00 Lunch Break
14:00 Lab practice on control
16:00 Break
16:30 Lab practice on distributed control
19:00 Banquet
21:00 Luminara tour (Candlelight Feast) in Pisa

Thursday, June 17

08:30 You can sleep longer
11:00 Scilab/Scicos (automatic code generation)
12:00 Project assignment
13:00 Lunch Break
14:00 Lab practice
16:00 Break
16:30 Lab practice
18:30 End of Session

Friday, June 18

08:30 Lab practice or Final Exam
10:30 Coffee Break
11:00 Project presentation and evaluations
13:00 Lunch Break
14:30 Closing remarks

Overview of Real-Time Scheduling

Giorgio Buttazzo

Scuola Superiore Sant'Anna, Pisa

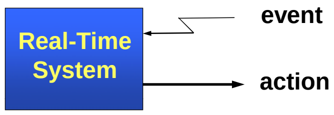
E-mail: buttazzo@sssup.it

Goal

Provide some background of RT theory that you can apply for implementing embedded control applications:

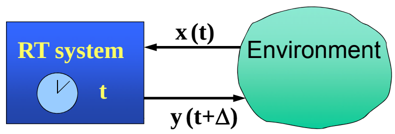
- Basic concepts and task models
- Periodic task scheduling
- Schedulability analysis
- Handling shared resources
- Accounting for blocking times

Real-Time system



A computing system able to respond to events within precise timing constraints.

Real-Time system



It is a system in which the correctness depends not only on the output values, but also on the **time** at which results are produced.

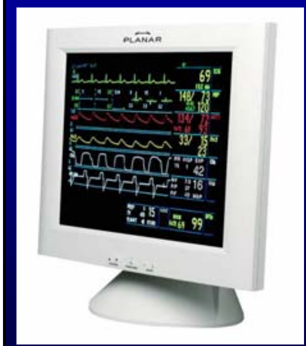
Flight control systems

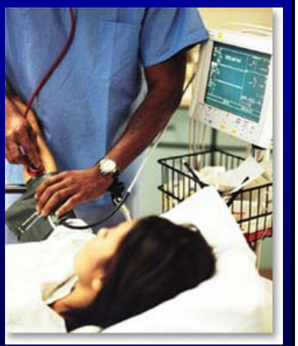
Flight simulators

Defense systems

Robotics

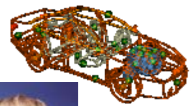
Medical monitoring Systems





… and many others

- automotive applications
- multimedia systems
- virtual reality
- small embedded devices
  - ⇒ cell phones
  - ⇒ digital cameras
  - ⇒ videogames
  - ⇒ smart toys

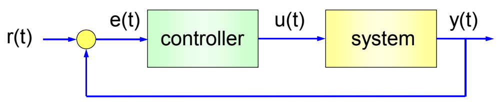


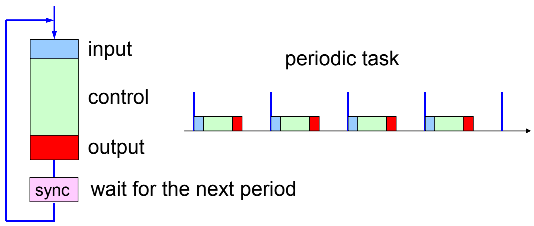
Interaction with the environment

- Timing constraints are imposed by the dynamics of the environment.
- The tight interaction with the environment requires the system to react to events within precise timing constraints.

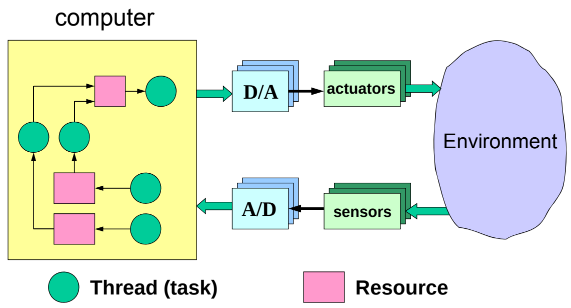
➡ The operating system is responsible for enforcing such constraints on task execution.

Implementing a control task

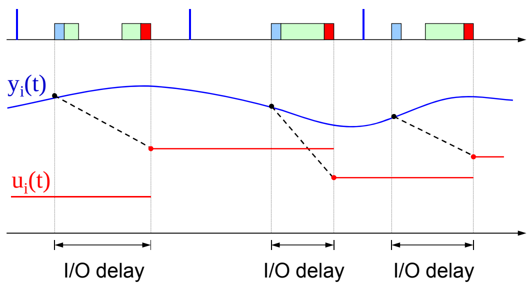




Software Vision

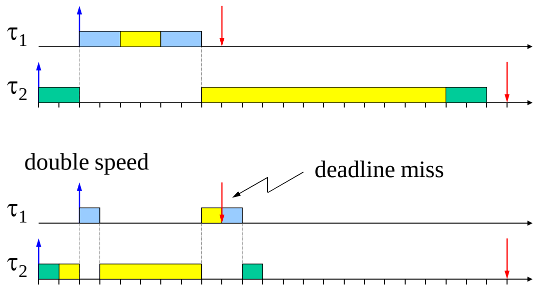


Kernel-induced delays



Other timing anomalies

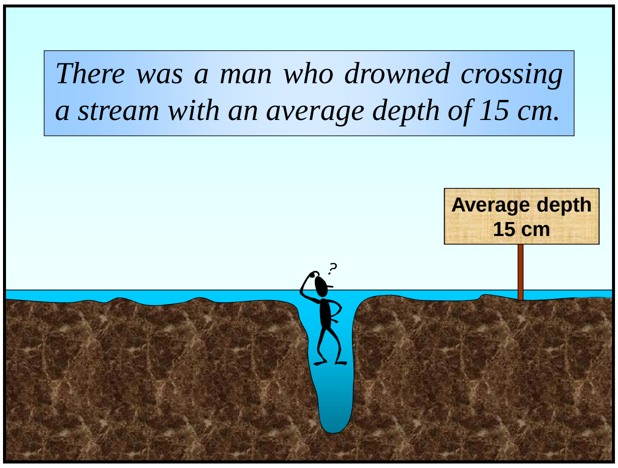
Real-Time ≠ Fast



Speed vs. Predictability

- The objective of a real-time system is to guarantee the timing behavior of each individual task.
- The objective of a fast system is to minimize the average response time of a task set. But …

Don't trust average when you have to guarantee individual performance



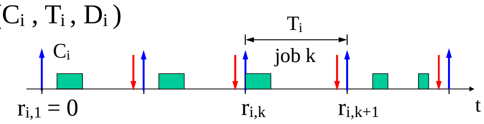
Achieving predictability

- The operating system is the part most responsible for a predictable behavior.
- Concurrency control must be enforced by:
  - ⇒ appropriate scheduling algorithms
  - ⇒ appropriate syncronization protocols
  - ⇒ efficient communication mechanisms
  - ⇒ predictable interrupt handling

Let's review the main scheduling results

Periodic task model

$\tau_i (C_i , T_i , D_i)$



For each periodic task, guarantee that:

- each job τ_{ik} is activated at $r_{ik} = (k-1)T_i$
- each job τ_{ik} completes within $d_{ik} = r_{ik} + D_i$

Priority Assignments

- **Rate Monotonic (RM):**

  $P_i \propto 1/T_i$ (static)

- **Deadline Monotonic (DM):**

  $P_i \propto 1/D_i$ (static)

- **Earliest Deadline First (EDF):**

  $P_i \propto 1/d_{ik}$ (dynamic) $d_{i,k} = r_{i,k} + D_i$

RM Optimality

RM is **optimal** among all fixed priority algorithms:

If there exists a fixed priority assignment which leads to a feasible schedule for Γ, then the RM assignment is feasible for Γ.

⇕

If Γ is not schedulable by RM, then it cannot be scheduled by any fixed priority assignment.

EDF Optimality

EDF is **optimal** among all algorithms:

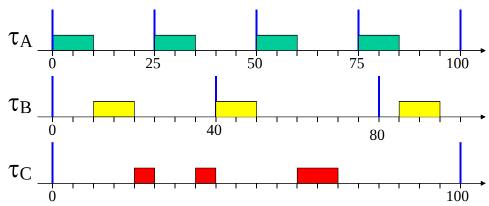
If there exists a feasible schedule for Γ, then EDF will generate a feasible schedule.

⇕

If Γ is not schedulable by EDF, then it cannot be scheduled by any algorithm.

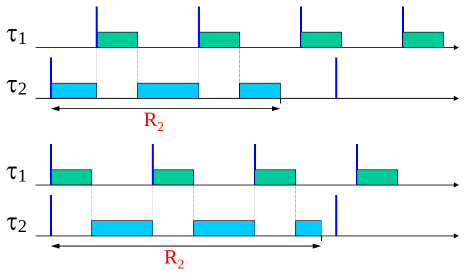
Rate Monotonic (RM)

- Each task is assigned a fixed priority P_i proportional to its rate.



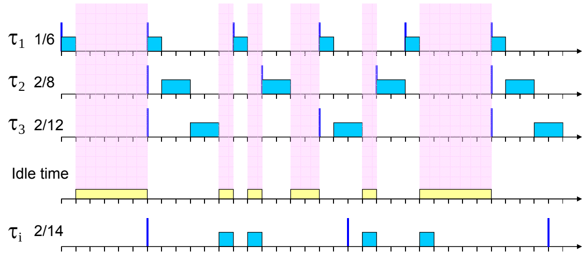
Critical Instant

For a task τ_i, it is the activation time that produces the longest response time.



Critical Instant

For independent preemptive tasks under fixed priorities, the critical instant of τ_i, occurs when it arrives together with all higher priority tasks.



How can we verify feasibility?

- Each task uses the processor for a fraction of time:

$$U_i = \frac{C_i}{T_i}$$

- Hence the total **processor utilization** is:

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

- U_p is a misure of the **processor load**

A necessary condition

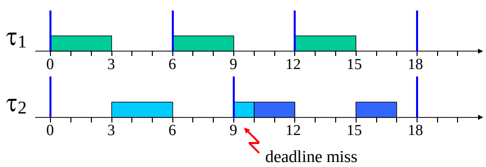
A necessary condition for having a feasible schedule is that $\mathbf{U_p \leq 1}$.

In fact, if $\mathbf{U_p > 1}$ the processor is overloaded hence the task set cannot be schedulable.

However, there are cases in which $\mathbf{U_p \leq 1}$ but the task set is not schedulable by RM.

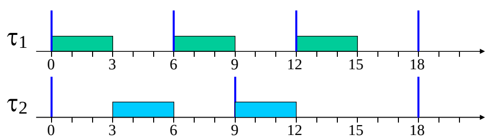
An unfeasible RM schedule

$$U_p = \frac{3}{6} + \frac{4}{9} = 0.944$$



Utilization upper bound

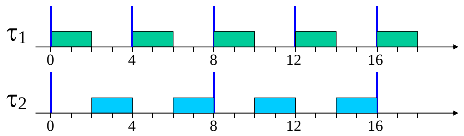
$$U_p = \frac{3}{6} + \frac{3}{9} = 0.833$$



NOTE: If C_1 or C_2 is increased, τ_2 will miss its deadline!

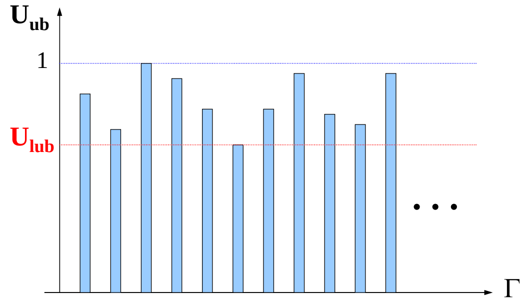
A different upper bound

$$U_p = \frac{2}{4} + \frac{4}{8} = 1$$



The upper bound U_{ub} depends on the specific task set.

The least upper bound



A sufficient condition

If $\mathbf{U_p} \leq \mathbf{U_{lub}}$ the task set is certainly schedulable with the RM algorithm.

NOTE

If $\mathbf{U_{lub}} < \mathbf{U_p} \leq \mathbf{1}$ we cannot say anything about the feasibility of that task set.

Basic results

In 1973, Liu & Layland proved that a set of ***n*** periodic tasks can be feasibly scheduled

under RM **if** $$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n\left(2^{1/n} - 1\right)$$

under EDF **if and only if** $$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$

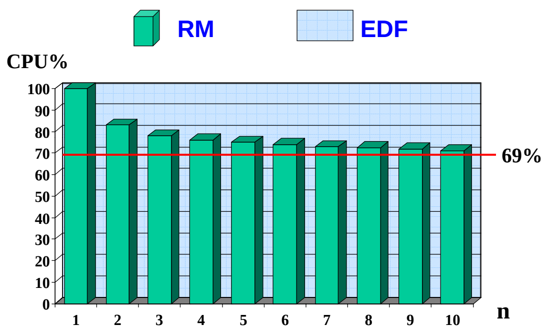
Assumptions:
- Independent tasks
- $\Phi_i = 0$
- $D_i = T_i$

Utilization bound for large *n*

$$U_{lub}^{RM} = n\left(2^{1/n} - 1\right)$$

for n → ∞ U_{lub} → *ln 2*

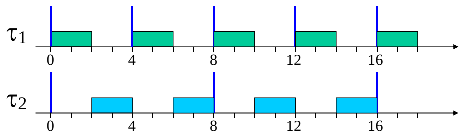
Schedulability bound



A special case

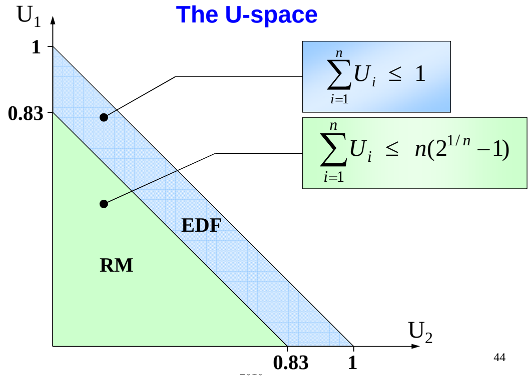
If tasks have harmonic periods U_{lub} = 1.

$$U_p = \frac{2}{4} + \frac{4}{8} = 1$$



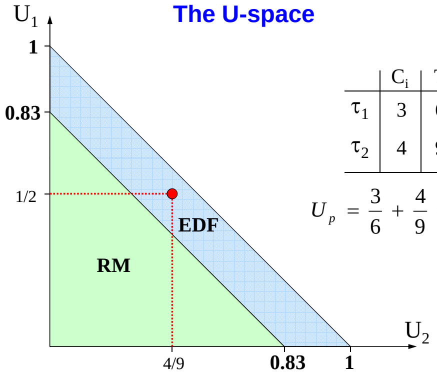
Schedulability region

The U-space



Schedulability region

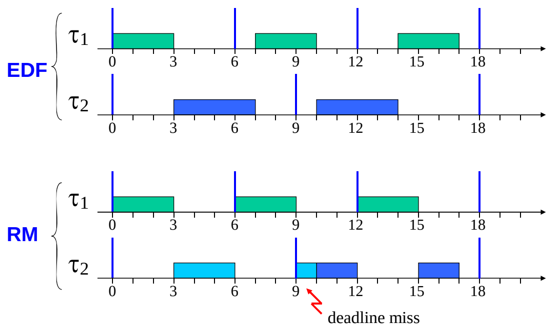
The U-space



	C_i	T_i
τ_1	3	6
τ_2	4	9

$$U_p = \frac{3}{6} + \frac{4}{9} = 0.94$$

Schedule



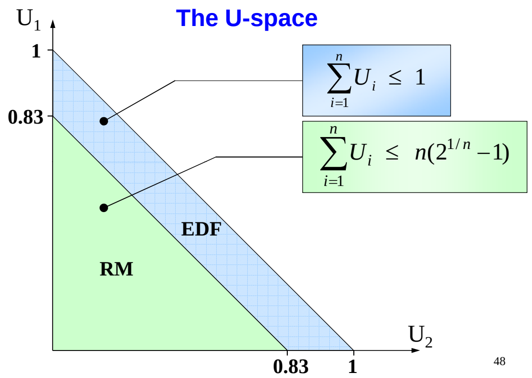
The Hyperbolic Bound

- In 2000, **Bini et al.** proved that a set of *n* periodic tasks is schedulable with RM if:

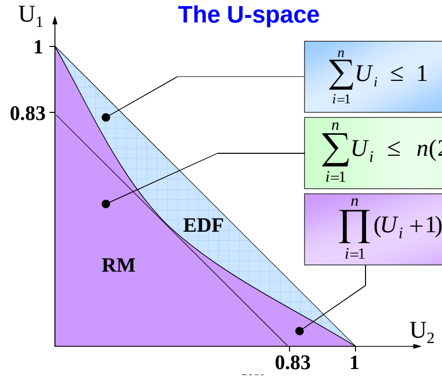
$$\prod_{i=1}^{n} (U_i + 1) \leq 2$$

Schedulability region

The U-space



Schedulability region



$$\sum_{i=1}^{n} U_i \le 1$$

$$\sum_{i=1}^{n} U_i \le n(2^{1/n} - 1)$$

$$\prod_{i=1}^{n} (U_i + 1) \le 2$$

49

Response Time Analysis
[Audsley '90]

- For each task τ_i compute the interference due to higher priority tasks:

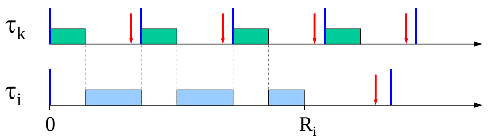
$$I_i = \sum_{P_k > P_i} C_k$$

- compute its response time as

$$R_i = C_i + I_i$$

- verify if $R_i \le D_i$

Computing the interference



Interference of τ_k on τ_i in the interval $[0, R_i]$:

$$I_{ik} = \left\lceil \frac{R_i}{T_k} \right\rceil C_k$$

Interference of high priority tasks on τ_i:

$$I_i = \sum_{k=1}^{i-1} \left\lceil \frac{R_i}{T_k} \right\rceil C_k$$

Computing the response time

$$R_i = C_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_i}{T_k} \right\rceil C_k$$

Iterative solution:

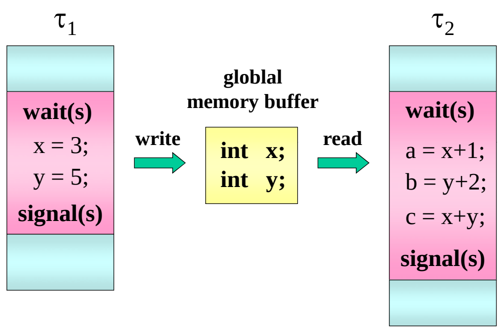
$$\begin{cases} R_i^0 = C_i \\ R_i^s = C_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_i^{(s-1)}}{T_k} \right\rceil C_k \end{cases}$$

iterate until $R_i^s > R_i^{(s-1)}$

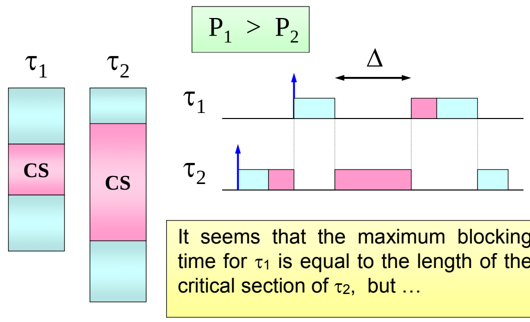
Handling shared resources

Problems caused by mutual exclusion

Critical sections

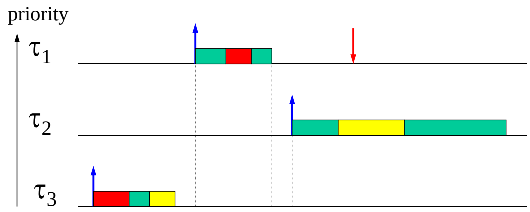


Blocking on a semaphore

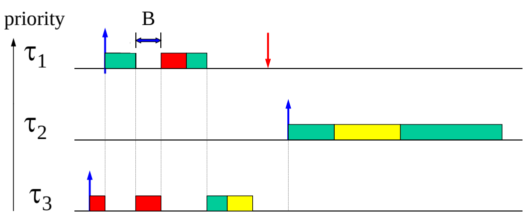


It seems that the maximum blocking time for τ_1 is equal to the length of the critical section of τ_2, but …

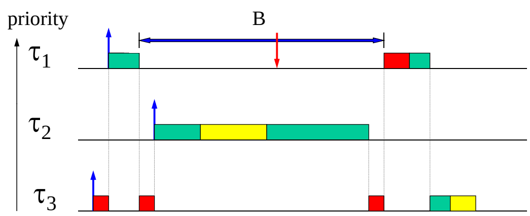
Schedule with no conflicts



Conflict on a critical section



Conflict on a critical section



Priority Inversion

A high priority task is blocked by a lower-priority task a for an unbounded interval of time.

Solution

Introduce a concurrency control protocol for accessing critical sections.

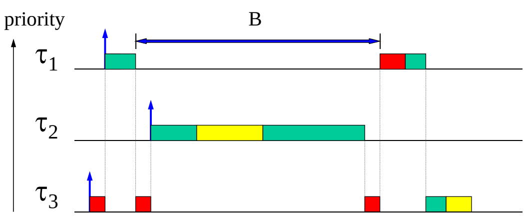
Non Preemptive Protocol

- Preemption is forbidden in critical sections.
- Implementation: when a task enters a CS, its priority is increased at the maximum value.

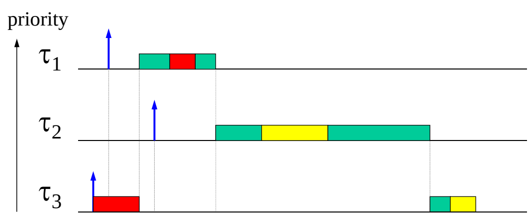
ADVANTAGES: simplicity

PROBLEMS: high priority tasks that do not use the same resources may also block

Conflict on critical section

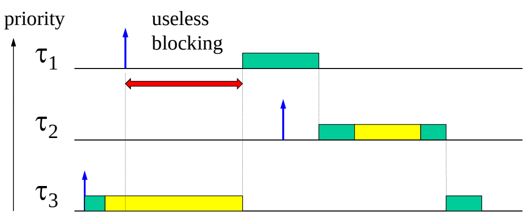


Schedule with NPP



$$P_{CS} = \max\{P_1, \ldots P_n\}$$

Problem with NPP



τ_1 cannot preemt, although it could

Highest Locker Priority (Immediate Priority Ceiling)

A task entering a resource R_k gets the highest priority among the tasks that use R_k

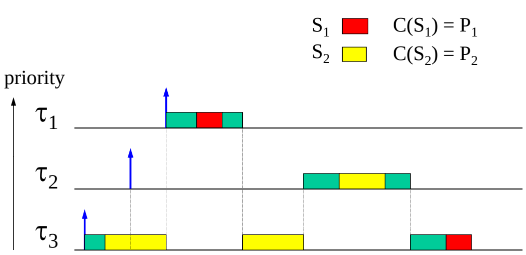
Implementation:

- Each task τ_i has a dynamic priority p_i initialized to P_i
- Each semaphore S_k has a ceiling

$$C(S_k) = \max\{P_i \mid \tau_i \text{ uses } S_k\}$$

- When τ_i locks S_k, p_i is increased to $C(S_k)$
- When τ_i unlocks S_k, its priority goes back to P_i

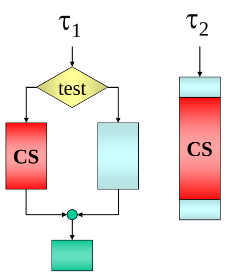
Schedule with HLP



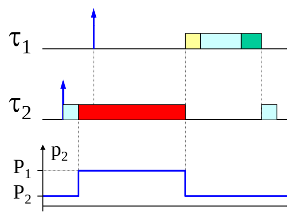
τ_2 is blocked, but τ_1 can preempt τ_3 within its critical section, because $P_1 > C(S_2)$

Problem with NPP and HLP

A task is blocked when attempting to preempt, not when accessing the resource.



τ_1 blocks just in case ...



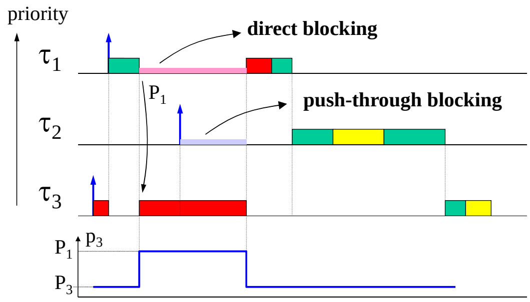
Priority Inheritance Protocol

[Sha, Rajkumar, Lehoczky, 90]

- A task increases its priority only if it blocks other tasks.
- A task τ_i in a resource R_k inherits the highest priority among those tasks it blocks.

$p_i(R_k) = \max \{P_h \mid \tau_h \text{ blocked on } R_k\}$

Schedule with PIP



Types of blocking

- **Direct blocking**

  A task blocks on a locked semaphore

- **Push-through blocking**

  A task blocks because a lower priority task inherited a higher priority.

BLOCKING:
a delay caused by a lower priority task

Identifying blocking resources

- A task τ_i can be blocked by those semaphores used by lower priority tasks
  - directly shared with τ_i (direct blocking)
  - shared with tasks having priority higher than τ_i (push-through blocking).

Theorem: τ_i can be blocked at most once by each of such semaphores

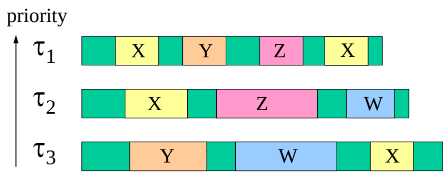
Theorem: τ_i can be blocked at most once by each lower priority task

Bounding blocking times

- Let $\mathbf{n_i}$ be the number of tasks with priority less than τ_i
- Let $\mathbf{m_i}$ be the number of semaphores that can block τ_i

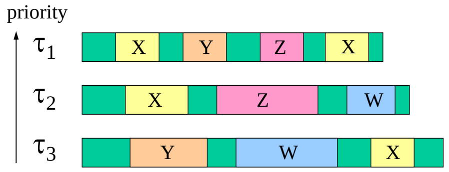
Theorem: τ_i can be blocked at most on the duration of $\alpha_i = \min(n_i, m_i)$ critical sections

Example



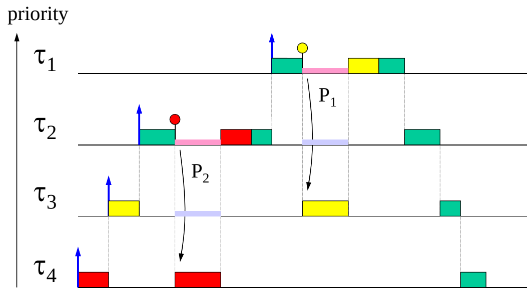
- τ_1 can be blocked once by τ_2 (on X_2 or Z_2) and once by τ_3 (on X_3 or Y_3)
- τ_2 can be blocked once by τ_3 (on X_3, Y_3 or W_3)
- τ_3 cannot be blocked

Example

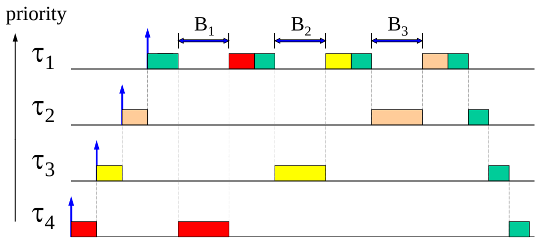


- $B_1 = \delta(Z_2) + \delta(Y_3)$
- $B_2 = \delta(W_3)$
- $B_3 = 0$

Schedule with PIP



Chained blocking with PIP

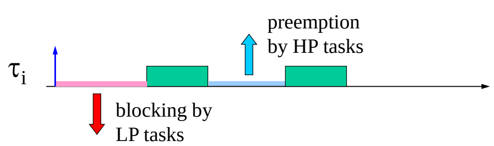


Theorem: τ_i can be blocked at most once by each lower priority task

Comparison

	NPP	HLP	PIP
# of blocking	1	1	α_i = min(n_i,m_i)
chained blocking	no	no	yes
deadlocks avoidance	yes	yes	no
pessimism	very high	high	low
transparency	yes	no	yes
stack sharing	yes	yes	no

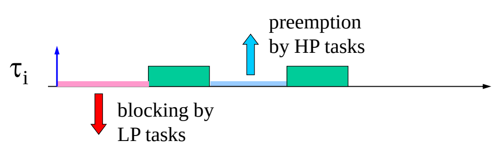
Accounting for blocking times



Utilization test

$$\forall i \quad \sum_{k=1}^{i-1}\frac{C_k}{T_k} + \frac{C_i+B_i}{T_i} \le i\left(2^{1/i}-1\right)$$

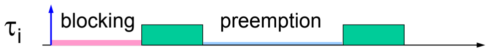
Accounting for blocking times



Hyperbolic bound

$$\forall i \quad \prod_{k=1}^{i-1}\left(\frac{C_k}{T_k}+1\right)\left(\frac{C_i+B_i}{T_i}+1\right) \le 2$$

Response Time Analysis



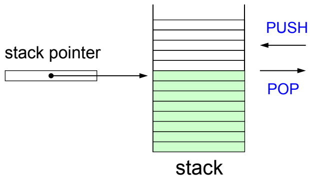
$$
\begin{cases}
R_i^0 = B_i + C_i \\
R_i^s = B_i + C_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_i^{(s-1)}}{T_k} \right\rceil C_k
\end{cases}
$$

iterate until $R_i^s > R_i^{(s-1)}$

Stack Sharing

Each task normally uses a private stack for

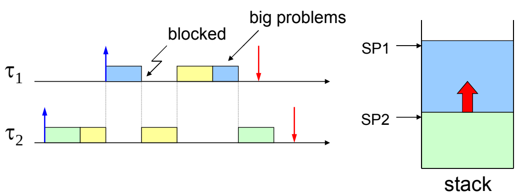
- saving context (register values)
- managing functions
- storing local variables



Stack Sharing

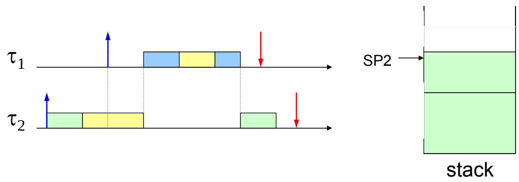
Why stack cannot be normally shared?

Suppose tasks share a resource: A



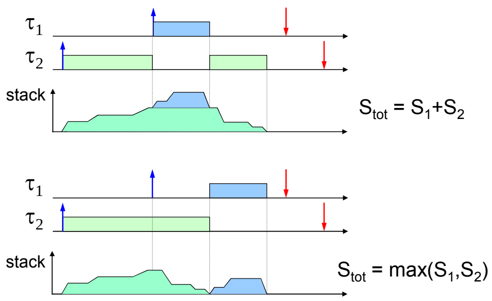
Stack Sharing

Why stack can be shared under NPP and HLP?



Saving stack memory

To save stack size, we should reduce preemption as much as possible:



Tasks grouping

To reduce preemption we can merge tasks into groups with the same priority level:

Consider 100 tasks, each with 10 Kb of stack

100 priorities → stack size = 1 Mb

10 groups of 10 tasks each → stack size = 100 Kb

stack saving = 90 %

Non preemptive scheduling

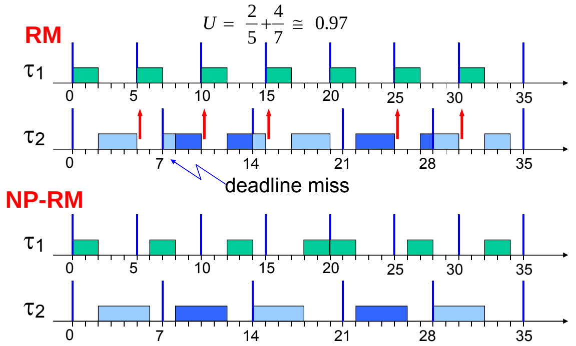
Advantages of NP scheduling

- It reduces context-switch overhead:
  - ➢ making WCETs smaller and more predictable.
- It simplifies the access to shared resources:
  - ➢ No semaphores are needed for critical sections
- It reduces stack size:
  - ➢ Task can share the same stack, since no more than one task can be in execution
- It allows achieving zero I/O Jitter:
  - ➢ finishing_time – start_time = C_i (constant)

Advantages of NP scheduling

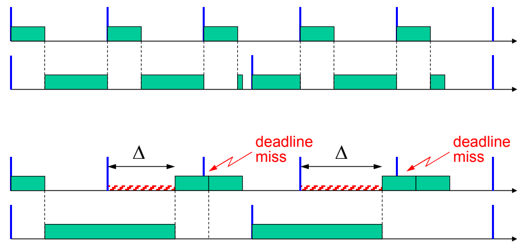
In fixed priority systems can improve schedulabiilty:

$$U = \frac{2}{5} + \frac{4}{7} \cong 0.97$$



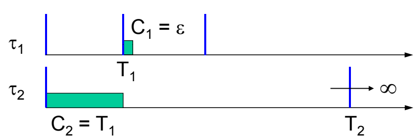
Disdvantages of NP scheduling

- In general, NP scheduling reduces schedulability introducing blocking delays in high priority tasks:



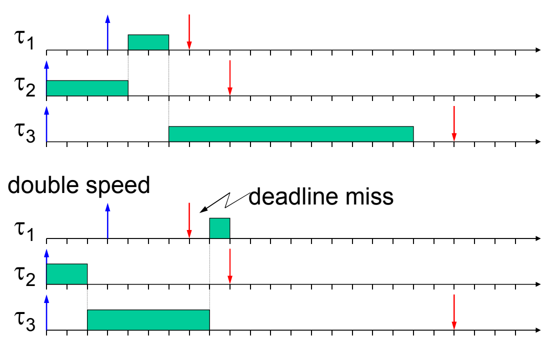
Disdvantages of NP scheduling

- The utilization bound under non preemptive scheduling drops to zero:



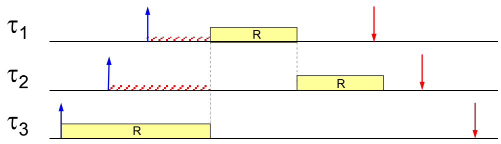
$$U = \frac{\varepsilon}{T_1} + \frac{C_2}{\infty} \longrightarrow 0$$

Non preemptive scheduling anomalies



Non-preemtive analysis

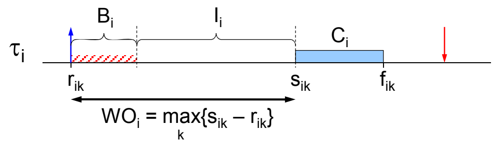
It is a special case of preemptive scheduling where all tasks share a single resource for their entire duration.



The max blocking time for task τ_i is given by the largest C_k among the lowest priority tasks:

$$B_i = \max\{C_j : P_j < P_i\}$$

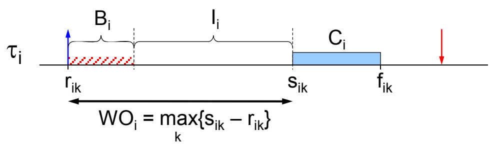
Response time analysis



WO_i = Worst-case occupied time: due to blocking from lp(i) and interference from hp(i)

NOTE: the end of WO_i cannot coincide with the activation of a higher priority task τ_h, which would increase WO_i by C_h

Response time analysis



$$WO_i = B_i + \sum_{k=1}^{i-1} \left(\left\lfloor \frac{WO_i}{T_k} \right\rfloor + 1 \right) C_k$$

$$R_i = WO_i + C_i$$

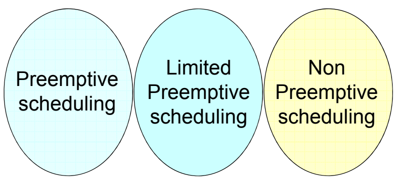
Response time analysis

$$\begin{cases} WO_i^{(0)} = B_i + \sum_{k=1}^{i} C_k \\ WO_i^{(s)} = B_i + \sum_{k=1}^{i-1} \left(\left\lfloor \frac{WO_i^{(s-1)}}{T_k} \right\rfloor + 1 \right) C_k \end{cases}$$

Stop when $WO_i^{(s)} = WO_i^{(s-1)}$

$$R_i = WO_i + C_i$$

Taking advantage of NP scheduling



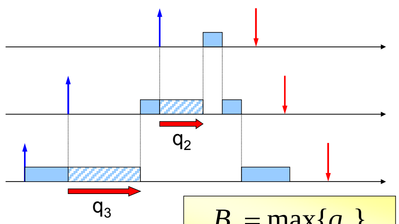
Trade-off solutions

- Deferred preemptions
- Fixed preemption points
- Preemption thresholds

Deferred Preemption

Each task can defer preemption up to q_i

NP regions are floating in the code (i.e., NP regions can start at any time)



$$B_i = \max_{P_j < P_i}\{q_j\}$$

Interesting problem

Given a preemptively feasible task set, reduce preemptions as much as possible still preserving schedulability.

→ Reducing context switch costs and WCETs

This means finding the longest non-preemptive chunk Q_i for each task that can still preserve schedulability.

- Under EDF → Baruah - ECRTS 2005
- Under Fix. Pr. → Yao et. al. - RTCSA 2009

A simple bound for Q_i

Q_i is related with the maximum blocking time that can be tolerated by higher priority tasks.

Let β_i be the maximum blocking time that can be tolerated by τ_i, called blocking tolerance.

Then, it must be $B_i \le \beta_i$

where $B_i = \max_{P_j < P_i}\{q_j\}$

Hence: $\max_{P_j < P_i}\{q_j\} \le \beta_i$

A simple bound for Q_i

$$\forall i \quad \max_{P_j < P_i}\{q_j\} \le \beta_i$$

$$\begin{cases} i = 1 & \max\{q_2, q_3, q_4\} \le \beta_1 \\ i = 2 & \max\{q_3, q_4\} \le \beta_2 \\ i = 3 & q_4 \le \beta_3 \end{cases}$$

$$\begin{cases} i = 1 & q_2 \le \beta_1 \\ i = 2 & q_3 \le \min\{\beta_1, \beta_2\} \\ i = 3 & q_4 \le \min\{\beta_1, \beta_2, \beta_3\} \end{cases}$$

A simple bound for Q_i

$$\begin{cases} i = 1 & Q_1 = \infty \\ i \ge 2 & Q_i = \min\{Q_{i-1}, \beta_{i-1}\} \end{cases}$$

Deriving β_i from the utilization test

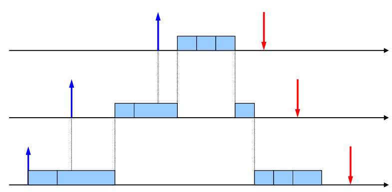
$$\left(\sum_{k=1}^{i} \frac{C_k}{T_k}\right) + \frac{B_i}{T_i} \le U_{\text{lub}}(i)$$

$$\left(\sum_{k=1}^{i} U_k\right) + \frac{\beta_i}{T_i} = U_{\text{lub}}(i)$$

$$\beta_i = T_i\left[U_{\text{lub}}(i) - \sum_{k=1}^{i} U_k\right]$$

Fixed Preemption Points (FPP)

- Each task τ_i is divided in m_i chunks: $q_{i,1}$ ... q_{i,m_i}
- It can only be preempted between chunks



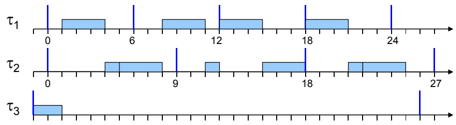
$$B_i = \max_{P_j < P_i}\{q_j^{\max}\}$$

Example

Let: τ_1 be fully non preemptive: $q_{11} = C_1 = 3$

τ_2 consisting of 2 NP chunks: $q_{21} = 1$, $q_{22} = 3$, $C_2 = 4$

τ_3 be fully non preemptive: $q_{31} = C_3 = 2$

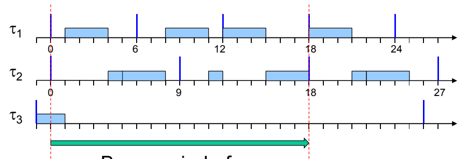


Note that:

- The worst case response time of τ_2 does not occur in the first instance.
- The interference on τ_2 is larger than $B_2 + C_1$.

Response Time Analysis (FPP)

- Must be carry out up to the busy period of each task.

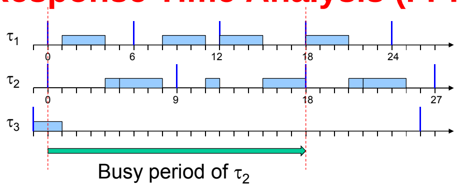


Busy period of τ_2

Level-i busy period

It is the interval in which the processor is busy executing tasks with priority higher than or equal to P_i, including blocking times.

Response Time Analysis (FPP)



Busy period of τ_2

Level-i busy period

It can be computed as the shortest interval that satisfies:

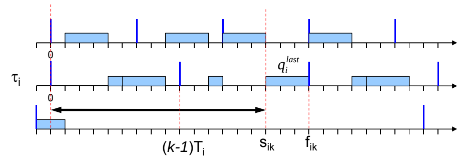
$$L_i = B_i + \sum_{h:P_h \geq P_i} \left\lceil \frac{L_i}{T_h} \right\rceil C_h$$

up to job N_i:

$$N_i = \left\lceil \frac{L_i}{T_i} \right\rceil$$

Initial value can be: $L_i^{(0)} = B_i + \sum_{h:P_h \geq P_i} C_h$

Response Time Analysis (FPP)



$$s_{ik} = B_i + (k-1)C_i + \left(C_i - q_i^{last}\right) + \sum_{h:P_h > P_i} \left(\left\lfloor \frac{s_{ik}}{T_h} \right\rfloor + 1 \right) C_h$$

$$f_{ik} = s_{ik} + q_i^{last}$$

$$R_{ik} = f_{ik} - (k-1)T_i$$

$$R_i = \max_{k \in [1, N_i]} \{R_{ik}\}$$

NOTE:

$$\begin{cases} s_{ik}^{(0)} = (k-1)T_i + \left(C_i - q_i^{last}\right) \\ N_i = \left\lceil \frac{L_i}{T_i} \right\rceil \end{cases}$$

Response Time Analysis (FPP)

for $(i=1$ to $n)$ {

$N_i = \lceil L_i / T_i \rceil$

$s_{ik}^{(0)} = (k-1)T_i + \left(C_i - q_i^{last}\right)$

$k = 1$

do {

$s_{ik} = B_i + (k-1)C_i + \left(C_i - q_i^{last}\right) + \sum_{h:P_h > P_i} \left(\left\lfloor \frac{s_{ik}}{T_h} \right\rfloor + 1 \right) C_h$

$f_{ik} = s_{ik} + q_i^{last}$

$R_{ik} = f_{ik} - (k-1)T_i$

if $(R_{ik} > R_i)$ **then** $R_i = R_{ik}$

if $(R_i > D_i)$ **then** *return(UNFEASIBLE)*

$k{+}{+}$

} **while** $(k \leq N_i)$

}

return(FEASIBLE)

Special cases

- Fully non preemptive scheduling

$$\begin{cases} q_i^{last} = C_i \\ B_i = \max_{P_j < P_i} \{C_j\} \end{cases}$$

- Deferred Preemption

$$\begin{cases} q_i^{last} = 0 \\ B_i = \max_{P_j < P_i} \{Q_j\} \end{cases}$$

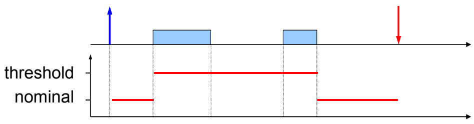
Preemption Thresholds (PT)

Each task has two priorities:

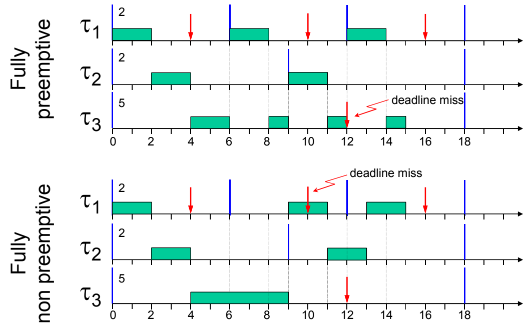
P_i nominal priority: used to enqueue the task in the ready queue and to preempt

θ_i threshold priority: used for task execution

threshold priority ≥ nominal priority

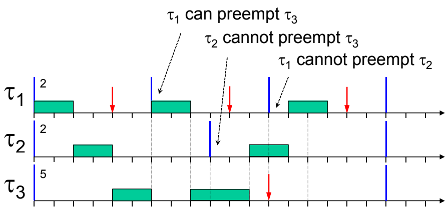


Unfeasible task set



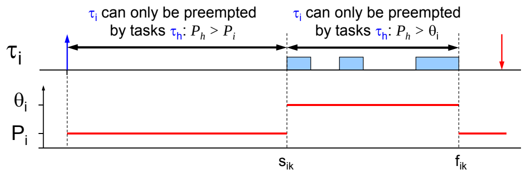
But feasible with preemption thresholds

P_i	θ_i
3	3
2	3
1	2



NOTE:
The same feasible schedule is obtained by splitting τ_3 in two non preemptive chuncks: $q_{31} = 2$, $q_{32} = 3$

Response time analysis (PT)



$$s_{ik} = B_i + (k-1)C_i + \sum_{h:P_h>P_i} \left(\left\lfloor \frac{s_{ik}}{T_h} \right\rfloor + 1 \right) C_h$$

$$f_{ik} = s_{ik} + C_i + \sum_{h:P_h>\theta_i} \left(\left\lceil \frac{f_{ik}}{T_h} \right\rceil - \left\lfloor \frac{s_{ik}}{T_h} \right\rfloor + 1 \right) C_h$$

Remarks

- Preemption Thresholds are easy to specify, but it is difficult to predict the number of preemptions and where they occur ⇒ large preemption overhead
- Deferred Preemption allows bounding the number of preemptions but it is difficult to predict where they occur. Note that the analysis assumes $q_i^{last} = 0$
- Fixed Preemption Points allow more control on preemptions and can be selected on purpose (e.g., to minimize overhead, stack size, and reduce WCETs).
- A large final chunk in τ_i reduces the interference from hp-tasks (hence R_i), but creates more blocking to hp-tasks.