Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya

Real-Time Systems

5-Resource access protocols

Antonio Camacho Santiago
antonio.camacho.santiago@upc.edu

Concepts and Definitions

Resource (R): SW structure needed to advance execution (memory, variable, registers, file…)

Private resource: resource dedicated to a particular process

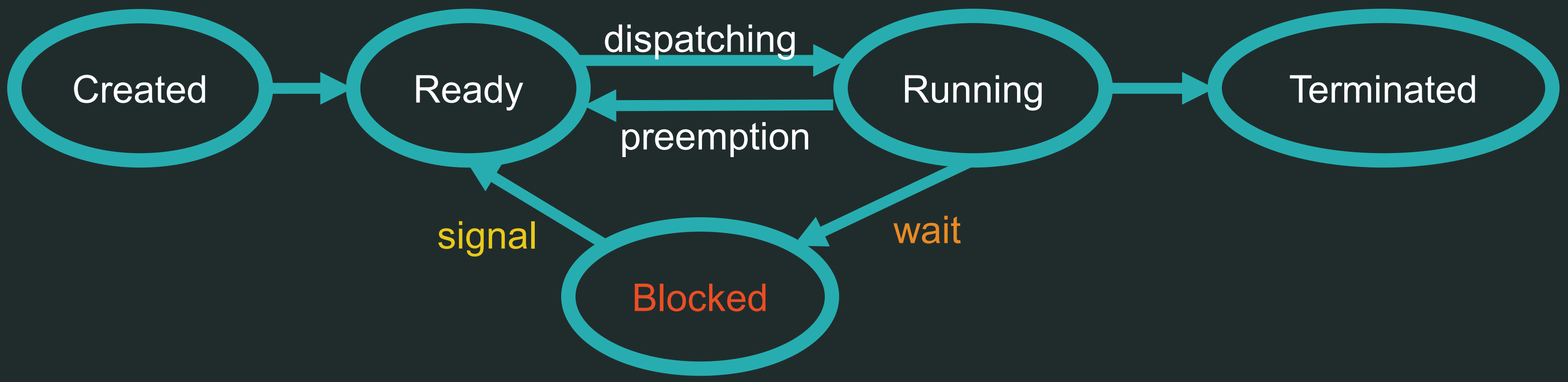
Shared resource: resource used by different tasks

Exclusive resource: resource protected against concurrent accesses

Mutual exclusion: resource access protocol to ensure consistency among competing tasks

Critical section (CS): piece of code executed under mutual exclusion

Semaphore (S): abstraction synchronization tool to build critical sections (wait→signal)



Problem statement

STATEMENT:
How entering a critical section to access an exclusive resource affects RT systems?

ATTENTION:
Tasks become blocked during critical sections affecting schedulability

PROBLEM:
Priority inversion phenomenon → A high priority task is blocked by a lower priority task

SOLUTION:
Resource Access Protocols:

- Non-Preemptive Protocol (NPP)
- Highest Locker Priority (HLP) or Immediate Priority Ceiling (IPC)
- Priority Inheritance Protocol (PIP)
- Priority Ceiling Protocol (PCP)
- Stack Resource Policy (SRP)

Priority inversion

What happens when no protocol exists to access resources?

Task 1 executes

Task 1 enters critical section and locks a semaphore

Task 2 has higher priority and preempts Task 1

Task 2 is blocked when trying to enter the critical section because the semaphore is locked

Task 1 releases semaphore and task 2 can enter the critical section

Task τ_i	Priority
τ_1	1
τ_2	2

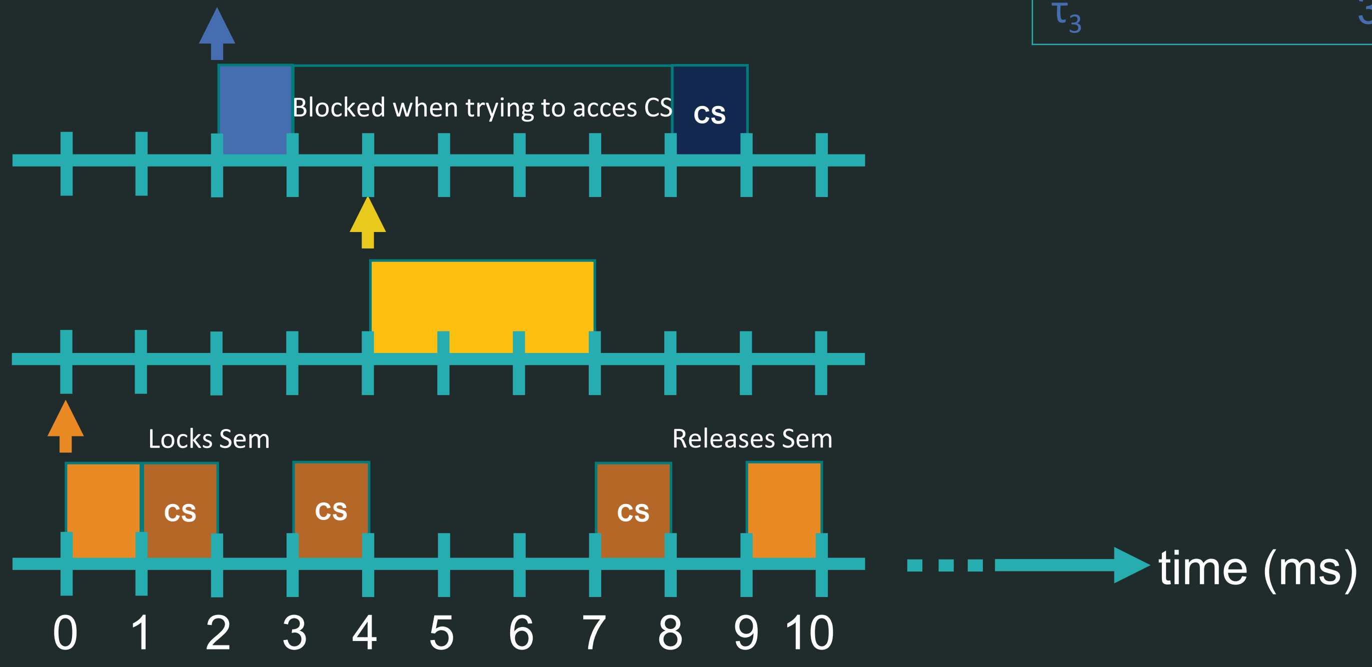
A low priority task blocks a high priority task. Blocking time must be bounded

Priority inversion

What happens when no protocol exists to access resources?

Similar to previous case, but task 3 is blocked for a longer time

Task τ_i	Priority
τ_1	1
τ_2	2
τ_3	3



A low priority task blocks a high priority task. Blocking time must be bounded

Pathfinder priority inversion

A low priority task locks a mutex when accessing a CS to put information into a MIL-STD-1553 bus

A medium priority task sends more data than expected causing more bus communications

The high priority task in charge of checking the bus, was blocked by the low priority task accessing CS

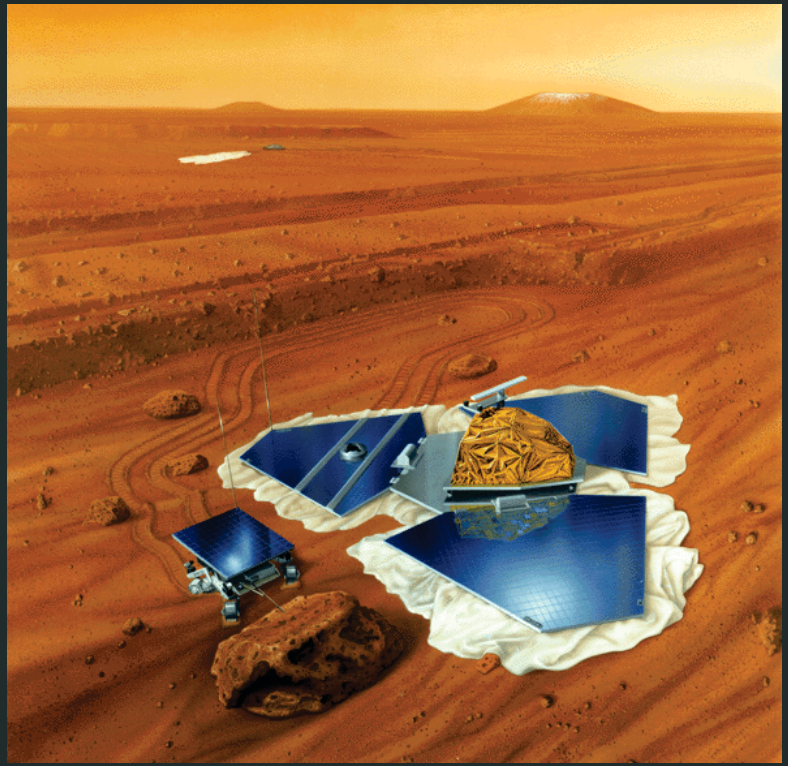
This causes the task to miss their deadlines

One of functionalities of this tasks that miss its deadline was to refresh the watchdog

The watchdog expires and causes multiple resets

Solution: Activate remotely a priority inheritance mechanism

Other explanations are also provided by researchers, but this one clearly shows the adverse effects of priority inversion



NON-PREEMTIVE PROTOCOL (NPP)

NPP rules

The approach for the NPP to avoid unbounded blocking time is to avoid preemption during the CS

Access Rule: a task blocks at the entrance of a CS, not at its activation time

Progress Rule: disable preemption when executing inside a CS by raising the priority

$$p_i(R_k) = \max_h \{P_k\}$$

```
wait(): p_i = max(P_1, P_2, ...)
```

Release Rule: enable preemption again by resetting the priority to the nominal value

```
signal(): p_i = P_i
```

Capital letters indicate nominal priority while small ones indicate dynamic priority

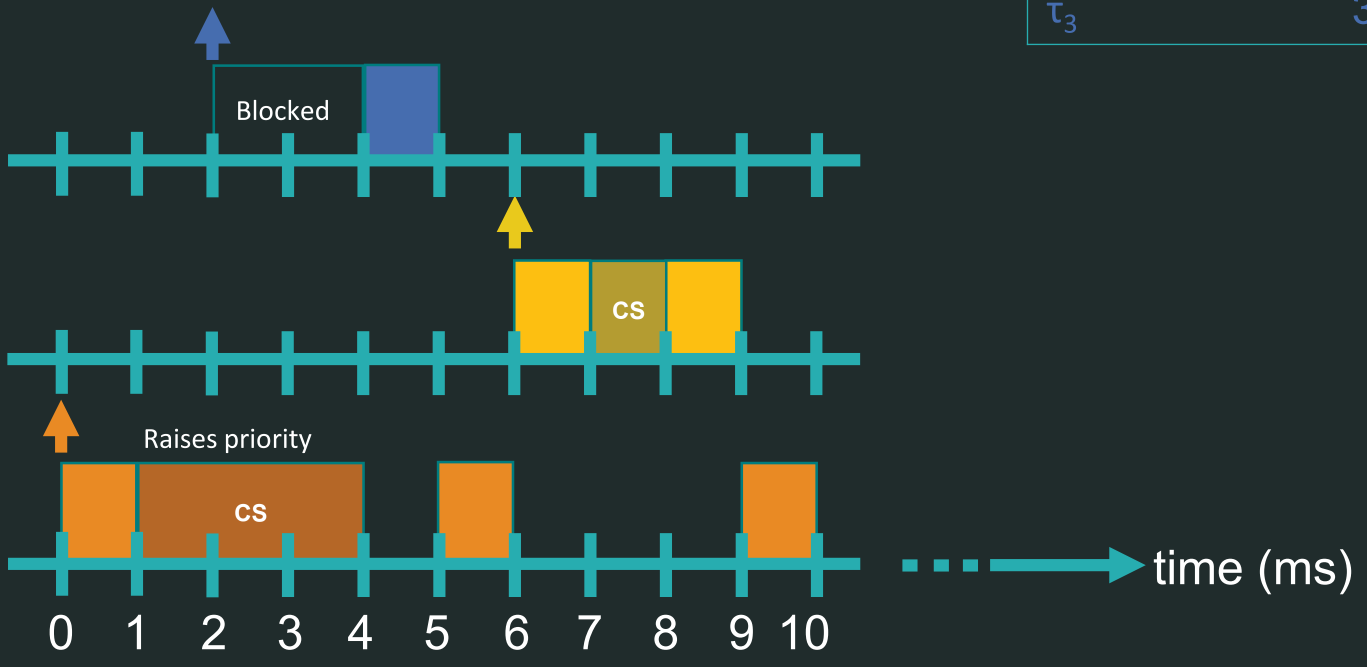
Recall: Priority inversion causes blocking time due to tasks accessing critical section in mutual exclusion. The blocking time must be bounded and the resource protocol tries to fix this issue

NPP example

Task 3 with highest priority is blocked during the CS of task 1

because the access protocol avoids preemption during CS

Task τ_i	Priority
τ_1	1
τ_2	2
τ_3	3



NPP blocking time

Maximum Blocking time (B_i) of a task τ_i implementing NPP corresponds to the duration of the longest CS of the lower priority tasks

$$B_i = \max_{j,k}\{\delta_{j,k} - 1 \mid Z_{j,k} \in \gamma_i\}$$

Note that a task τ_i cannot preempt a lower priority task τ_j when τ_j is inside a CS

Thus τ_i can be blocked by any CS of a lower priority task

$$\gamma_i = \{Z_{j,k} \mid P_j < P_i \ , k = 1, \ldots, m\}$$

where:

γ_i is the set of all longest CS that can block task τ_i

B_i is the maximum blocking time that task τ_i can suffer

$Z_{j,k}$ is the longest CS of task τ_j guarded by semaphore S_k

P_i is the priority of task τ_i

$\delta_{j,k}$ is the duration of $Z_{j,k}$

NPP summary

Pros:

- Solves the priority inversion phenomenon
- Simple
- Efficient
- Each task can block at most on a single critical section.
- No deadlocks

Cons:

- High priority tasks are blocked by low priority tasks
- Task may be blocked even if they do not use any resource
- If the critical section is very time consuming, the higher priority tasks are blocked for a long time

HIGHEST LOCKER PRIORITY (HLP)

HLP rules

Similar to NPP but the priority is increased until the maximum value of the tasks sharing the resource

Highest Locker Priority (HLP) is also known as Immediate Priority Ceiling (IPC)

Access Rule: a task blocks at the entrance of a CS, not at its activation time

Progress Rule: raise the priority to the value of the higher priority task that uses the resource

$p_i(R_k) = \max_h \{P_k \mid \tau_h \text{ uses } R_k\}$

```
C(R) ≝ ceiling of resource R
wait(): pi(R)= C(R)
```

Release Rule: enable preemption again by resetting the priority to the nominal value

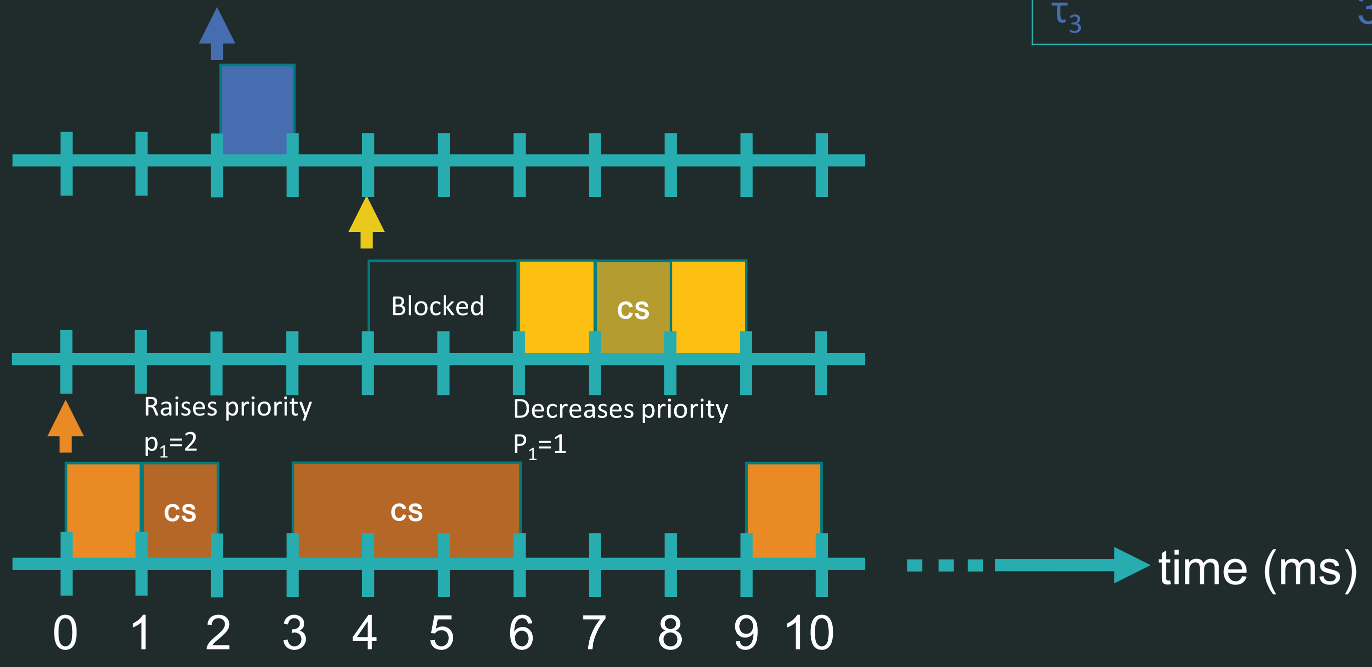
```
signal(): pi = Pi
```

HLP example

Task 1 and 2 share a resource *R* accessing a CS

Even though task's 1 priority is low, it is increased until the ceiling value of the priorities accessing *R* (i.e. P=2)

Task τ_i	Priority
τ_1	1
τ_2	2
τ_3	3



HLP blocking time

Maximum Blocking time (B_i) of a task τ_i implementing HLP corresponds to the duration of the longest CS among those that can block τ_i

$$B_i = \max_{j,k}\{\delta_{j,k} - 1 \mid Z_{j,k} \in \gamma_i\}$$

A task can be blocked at most once, for the duration of a single CS belonging to lower priority tasks with a resource ceiling higher or equal than its priority

$$\gamma_i = \{Z_{j,k} \mid P_j < P_i \text{ and } C(R_k) \geq P_i\}$$

Note:

γ_i is the set of all longest CS that can block task τ_i

B_i is the maximum blocking time that task τ_i can suffer

$Z_{j,k}$ is the longest CS of task τ_j guarded by semaphore S_k

P_i is the priority of task τ_i

$\delta_{j,k}$ is the duration of $Z_{j,k}$

HLP summary

Pros:

Solves the priority inversion phenomenon
Simple
Efficient
Each task can block at most on a single critical section.
No deadlocks
Stack sharing
A little improvement compared to NPP: Task cannot be blocked if they do not use any resource

Cons:

High priority tasks are blocked by low priority tasks
If the critical section is very time consuming, the higher priority tasks requiring the resource is blocked for a long time

PRIORITY INHERITANCE PROTOCOL (PIP)

PIP rules

When a task blocks higher priority tasks, it inherits the priority of the blocked tasks (even if they do not use the resource)

Access Rule: a task blocks at the entrance of a CS if the resource is locked

Progress Rule: inside resource *R*, a task executes with the highest priority of the tasks blocked on *R*.

$$p_i(R_k) = \max_h \{P_k \mid \tau_h \text{ blocked on } R_k\}$$

Release Rule: At exit, the dynamic priority of the task is reset to its nominal priority P_i

PIP implementation

Implement $p_i(R_k) = \max_h\{P_k \mid \tau_h \text{ blocked on } R_k\}$

```
wait(s):
if (s==0)
{
    suspend the task exe in the semaphore queue
    find the task that locks the semaphore
    pk = Pexe
    scheduler
}else{
    s=0
}
```

```
signal(s):
if (there are blocked tasks)
{
    awake the highest priority tasks in the semaphore queue
    pexe = Pexe
}else{
    s=1
}
```

PIP example 1

Normal scheduling until t=3

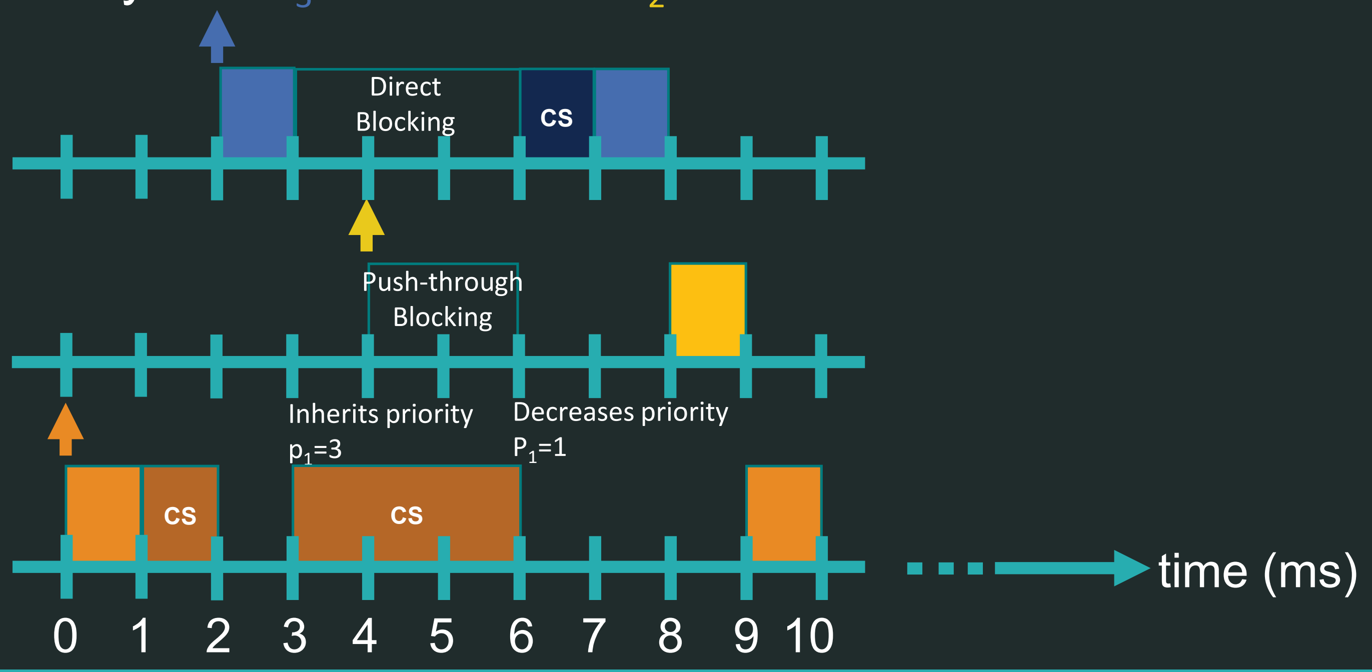
At t=3, τ_1 blocks τ_3 due to CS and τ_1 inherits the priority of τ_3

At t=4, τ_2 cannot preempt τ_1 because $P_1=3>2=P_2$

At t=6, τ_1 exists CS and recovers its original priority, also τ_3 is resumed

At t=8, the highest priority task τ_3 finishes and τ_2 starts

Task τ_i	Priority
τ_1	1
τ_2	2
τ_3	3

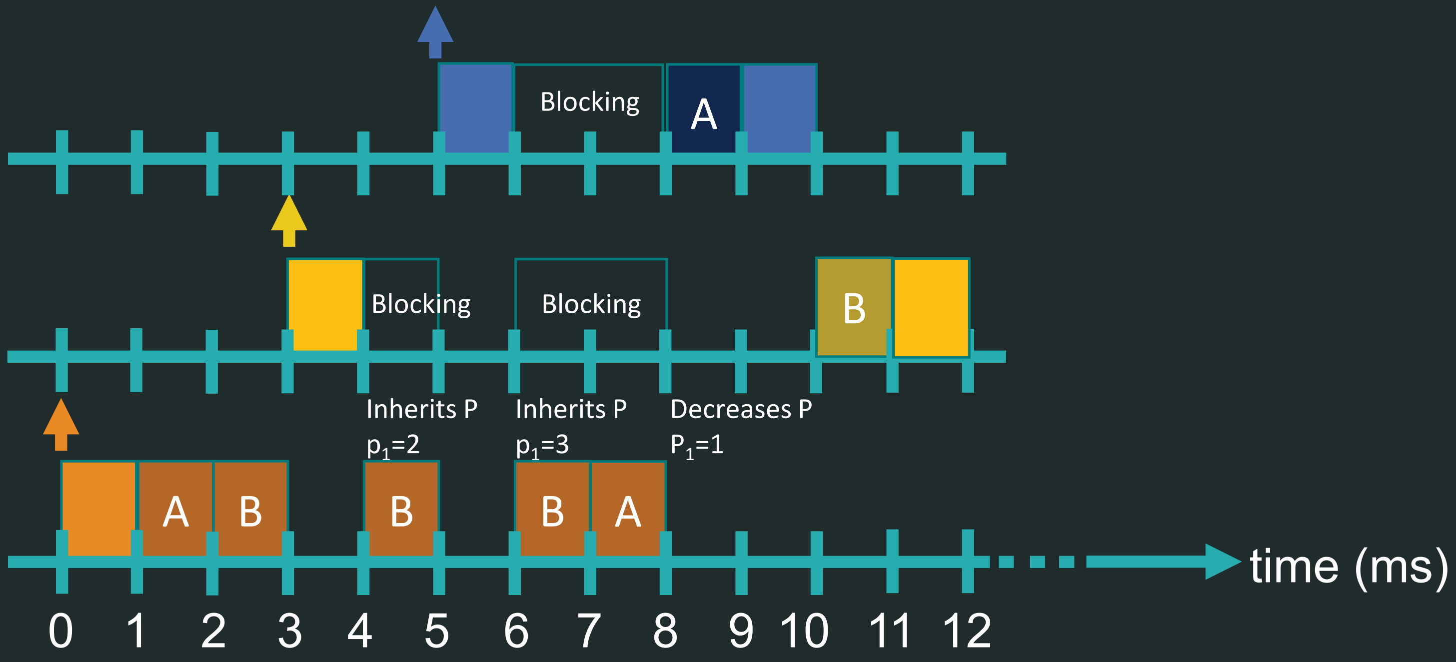


PIP example 2

Three tasks with normal and critical sections execution

Two resources A and B following a sequence for each task

Task τ_i	Priority	Sequence
τ_1	1	EABBBA
τ_2	2	EBE
τ_3	3	EAE



PIP blocking time

Maximum Blocking time (B_i) of a task τ_i implementing PIP

$$B_i = \min\{B_i^I, B_i^S\}$$

where:

$$B_i^I = \sum_{j=i+1}^{n} \max_k \{\delta_{j,k} - 1 \mid C(S_k) \geq P_i\} \qquad B_i^S = \sum_{k=1}^{m} \max_{j>i} \{\delta_{j,k} - 1 \mid C(S_k) \geq P_i\}$$

$$C(S_k) \stackrel{\text{def}}{=} \max_i \{P_i \mid S_k \in \sigma_i\}$$

Thus τ_i can be blocked by any CS sharing resource *R* of a lower priority task

Note:

σ_i is the set of semaphores used by τ_i

B_i is the maximum blocking time that task τ_i can suffer

$Z_{j,k}$ is the longest CS of task τ_j guarded by semaphore S_k

P_i is the priority of task τ_i

$\delta_{j,k}$ is the duration of $Z_{j,k}$

PIP summary

Pros:

- Solves the priority inversion phenomenon
- Tasks are blocked when needed (as opposed to NPP and HLP which blocks at activation)

Cons:

- Chained block
- Deadlock (problem with nested CS)
- Complexity

PRIORITY CEILING PROTOCOL (PCP)

PCP rules

Priority Ceiling Protocol can be viewed as a PIP + access test to avoid chained blocking and deadlocks

Access Rule: a task can access a resource if it passes the PCP access test

Progress Rule: inside resource R, a task executes with the highest priority of the tasks blocked on R.

$$p_i(R_k) = \max_h\{P_k \mid \tau_h \text{ blocked on } R_k\}$$

Release Rule: At exit, the dynamic priority of the task is reset to its nominal priority P_i.

PCP access test

The access test does not allow a task to enter a CS if there are locked semaphores than can block it. Therefore, once a task enters its first CS, it can never be blocked by lower priority tasks

Each resource is assigned a resource ceiling

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

PCP access test:

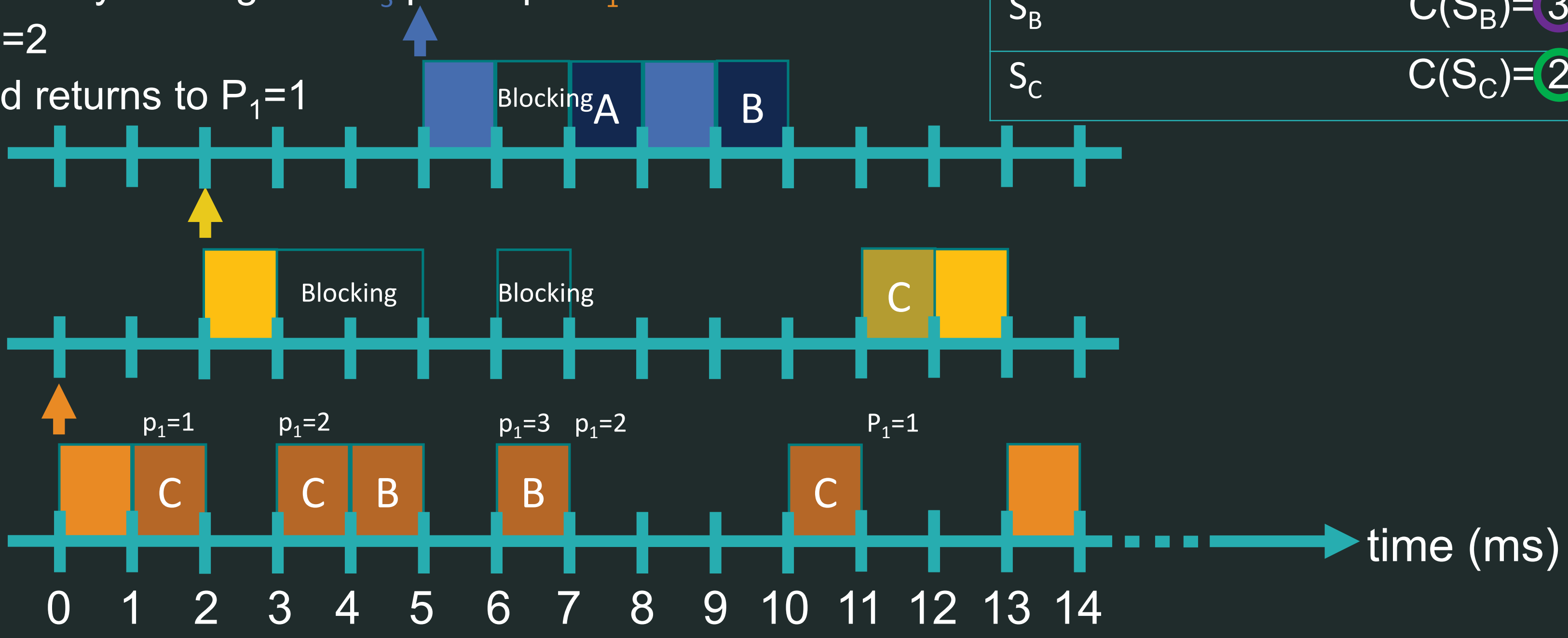
$$P_i > \max\{C(S_k)\colon\ s_k \text{ locked by tasks } \neq \tau_i\}$$

The access test states that any task can enter a CS only when its priority is higher than the maximum ceiling of the locked semaphores

PCP example

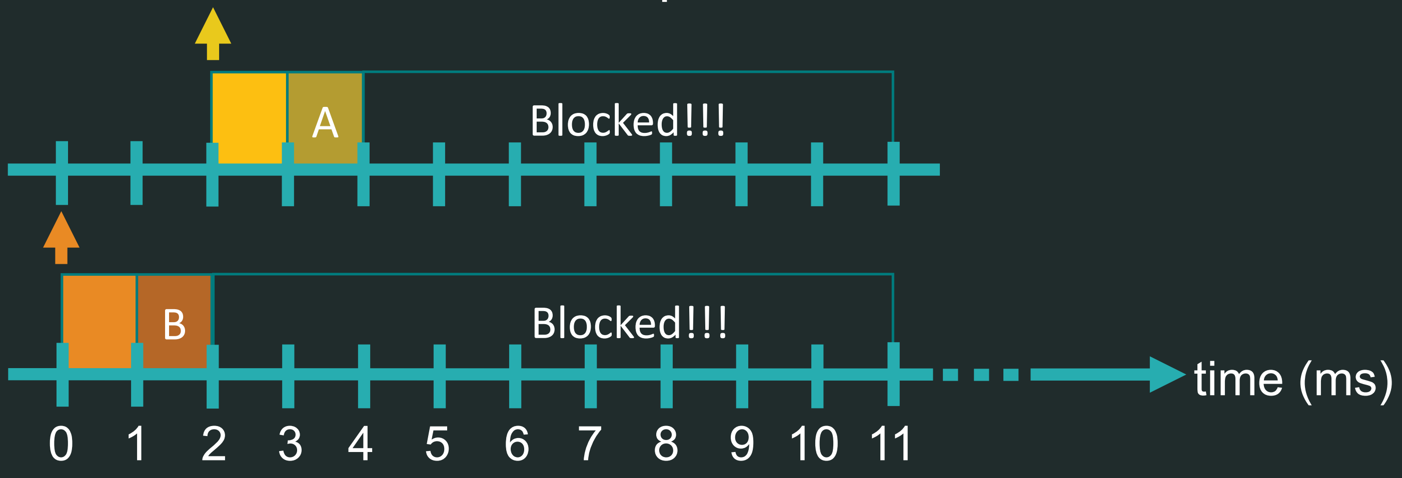
At t=1, τ_1 locks S_c . At t=2 τ_2 preempts τ_1
At t=3, τ_2 tries to lock S_C but it is blocked by PCP since $P_2 \leq C(S_C)$.
At t=3, τ_1 inherits the priority of τ_2 and continues
At t=4, τ_1 enters the nested CS and locks S_B (no one has lock S_B before)
At t=5, τ_3 activates and preempts τ_1 because $P_3=3>2=p_1$
At t=6, τ_3 tries to lock S_A but cannot because $P_3=3\leq 3= C(S_B)$
Then , τ_1 inherits the priority of τ_3
At t=7, τ_1 releases S_B and returns to priority $p_1=2$ due to S_C.
Also, $P_3=3>2=C(S_c)$ “locked by nesting” and τ_3 preempts τ_1
At t=10, τ_1 resumes at $p_1=2$
At t=11, τ_1 unlocks S_C and returns to $P_1=1$

Task τ_i	Priority	Sequence
τ_1	1	ECCBBCE
τ_2	2	ECE
τ_3	3	EAEB
Semaphore		**Semaphore ceiling**
S_A		$C(S_A)=3$
S_B		$C(S_B)=3$
S_C		$C(S_C)=2$

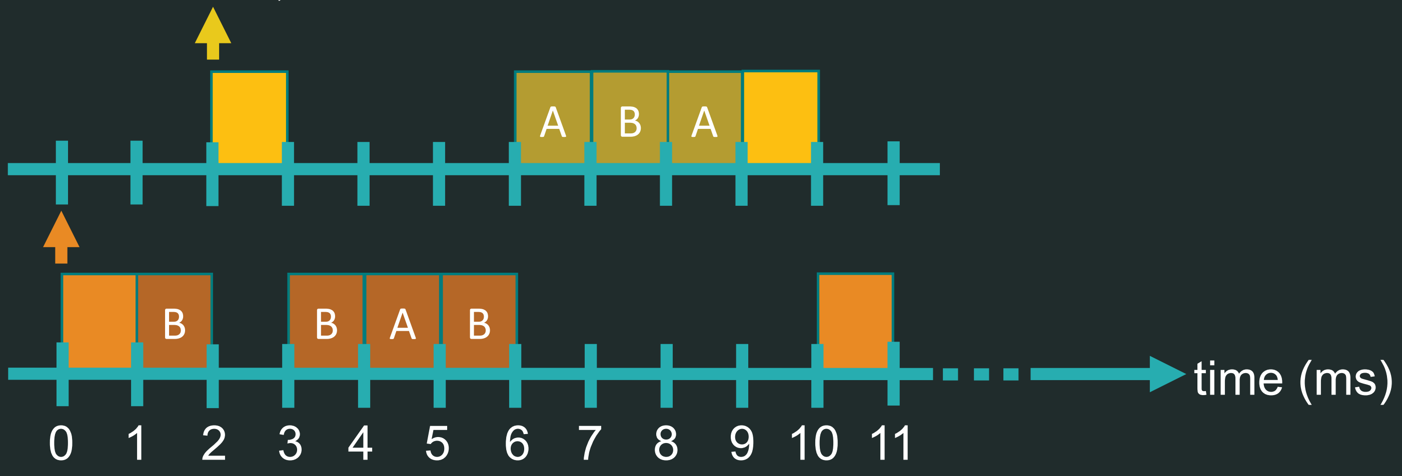


PCP avoids deadlock

Without PCP, deadlock is possible in nested CS



With PCP, deadlock is avoided



Task τ_i	Priority	Sequence
τ_1	1	EBBABE
τ_2	2	EABAE

Semaphore	Semaphore ceiling
S_A	$C(S_A)=2$
S_B	$C(S_B)=2$

PCP blocking time

Maximum Blocking time (B_i) of a task τ_i implementing PCP

The blocking time corresponds to the duration of the longest CS among those that can block the task

$$B_i = \max_{j,k}\{\delta_{j,k} - 1 \mid Z_{j,k} \in \gamma_i\}$$

where:

$$\gamma_i = \{Z_{j,k} \mid P_j < P_i \text{ and } C(R_k) \geq P_i\}$$

Thus τ_i can be blocked by CS belonging to lower priority tasks with a resource ceiling higher than or equal to P_i

Note:

γ_i is the set of all longest CS that can block task τ_i

B_i is the maximum blocking time that task τ_i can suffer

$Z_{j,k}$ is the longest CS of task τ_j guarded by semaphore S_k

P_i is the priority of task τ_i

$\delta_{j,k}$ is the duration of $Z_{j,k}$

PCP summary

Pros:

- Solves the priority inversion phenomenon
- Avoids chained blocking since blocking lasts the length of a critical section
- Avoids deadlocks when nesting critical sections

Cons:

- Unnecessary blocking
- Complexity

STACK RESOURCE POLICY (SRP)

SRP rules

Stack Resource Policy is a modification of PCP which can be used under EDF schedulers (based on dynamic priorities)

Tasks have a dynamic or fixed priority p_i, a preemption level π_i which determines at design stage if a task can preempt another task, and a value of how many units of resource R_k are used by task τ_i

Resources are characterized by a maximum number of units N_k and actual available units n_k

Dynamic ceiling is the highest preemption level of tasks that can block on R_k

$$C_R(n_k) = \max\{0, \pi_i : n(R_k) < \mu_i(R_k)\}$$

SRP preemption rule: A ready task τ_i can preempt the task being executed τ_{exe} if and only if

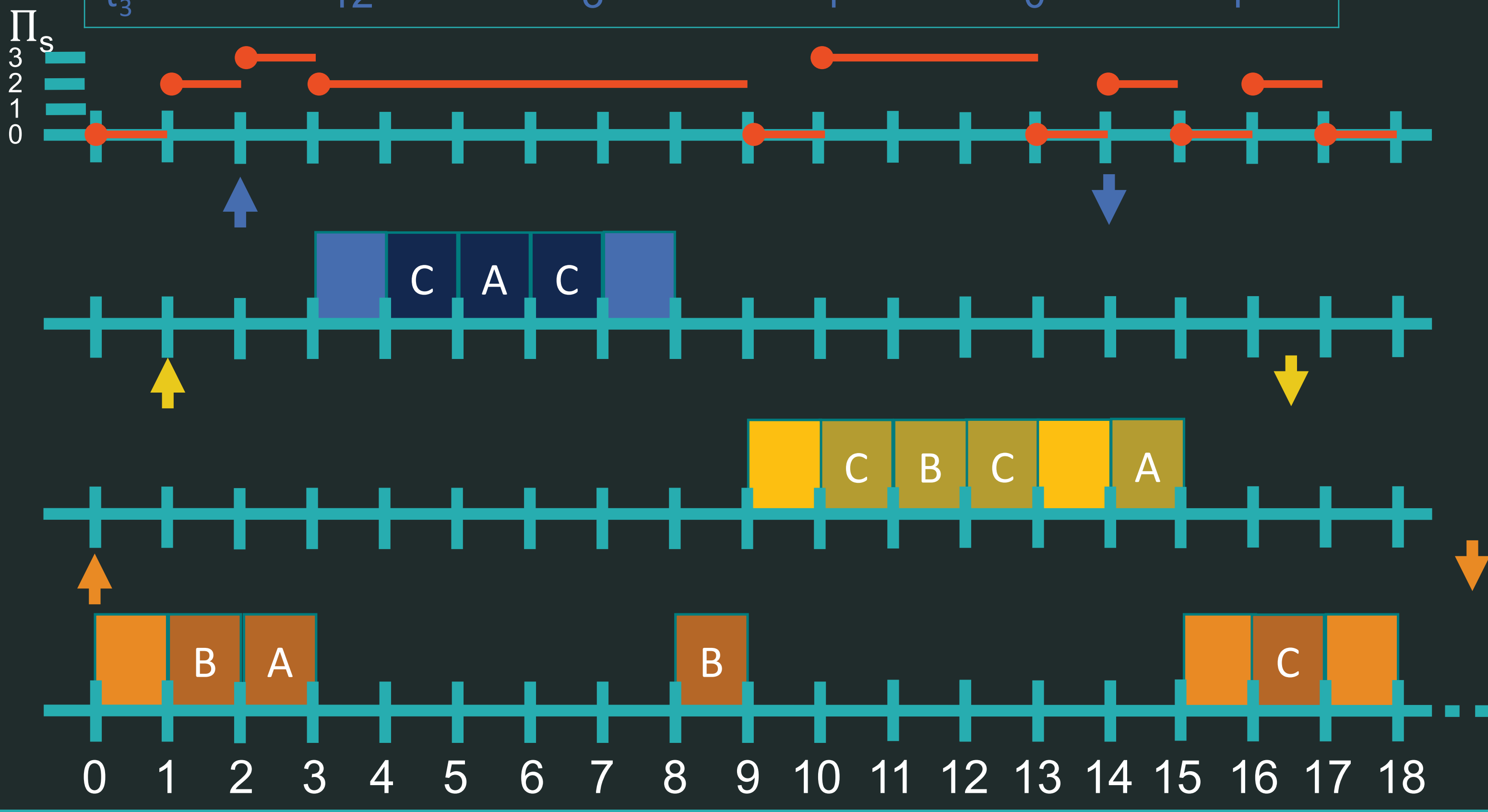
$$p_i > P_{exe} \text{ and } \pi_i > \Pi_s$$

where $\Pi_s = \max\{C_R(n_k)\}$ is the system ceiling

SRP example

Task τ_i	Deadline	Preemption level π_i	$\mu_i(R_A)$	$\mu_i(R_B)$	$\mu_i(R_C)$
τ_1	20	1	3	1	1
τ_2	16	2	2	1	3
τ_3	12	3	1	0	1

	$C_R(3)$	$C_R(2)$	$C_R(1)$	$C_R(0)$
R_A	0	1	2	3
R_B	-	-	0	2
R_C	0	2	2	3



time (ms)

$\mu_i(R_x)$→maximum number of units that can be simultaneously requested by τ_i to R_k computed offline by code inspection

$C_R(R_x)$→resource ceiling based on the highest preemption level of the tasks that could be blocked on R_k

SRP example explanation

At t=0, τ_1 starts. The resource ceiling is zero because all resources are available

At t=1, τ_1 takes the only unit of R_B. The system ceiling is set to $\Pi_s = 2$ which is the highest preemption level of tasks using R

Then, τ_1 blocks τ_2

At t=2, τ_1 takes all the units of R_A. The system ceiling is set to $\Pi_s = 3$ which is the highest preemption level of tasks using R

Then, τ_1 blocks τ_3

At t=3, τ_1 releases R_A, the system ceiling is $\Pi_s = 2$. Then τ_3 preempts τ_1

Between t=3 and t=8, no one can preempt τ_3 because of its preemption level

At t=8, τ_3 finishes and τ_1 resumes

At t=9, τ_1 releases R_B, and the system ceiling comes back to zero. Then blocks τ_2 can preempt τ_1

Between t=9 and t=15, no one can preempt τ_2 because of its preemption level and all the resources needed are available

SRP blocking time

The blocking time corresponds to the duration of the longest CS among those that can block the task

Maximum Blocking time (B_i) of a task τ_i implementing SRP

$$B_i = \max_{j,k}\{\delta_{j,k} - 1 \,|Z_{j,k} \in \gamma_i\}$$

where:

$$\gamma_i = \{Z_{j,k} \mid \pi_j < \pi_i \text{ and } C_{S_k}(0) \geq \pi_i\}$$

Note that:

γ_i is the set of all longest CS that can block task τ_i

B_i is the maximum blocking time that task τ_i can suffer

$Z_{j,k}$ is the longest CS of task τ_j guarded by semaphore S_k

π_i is the preemption level of task τ_i

$\delta_{j,k}$ is the duration of $Z_{j,k}$

SRP summary

Pros:

- Solves the priority inversion phenomenon
- Specifically designed for EDF (the previous resource access protocols where based on fixed priorities schedulers, RM or DM although they can be also adapted to EDF)
- Avoids chained blocking since blocking lasts the length of a critical section
- Avoids deadlocks when nesting critical sections
- Each task can be blocked at most once
- Tasks block when attempting to preempt
- Task stack space can be shared!!!

Cons:

- Unnecessary blocking
- Hard to understand, but easy to implement

SCHEDULABILITY OF RESOURCE ACCESS PROTOCOLS

Schedulability of Resource Access Protocols

RM:

Utilization factor:

$$\sum_{h:P_h>P_i} \frac{C_h}{T_h} + \frac{C_i + B_i}{T_i} \leq i\left(2^{1/i} - 1\right) \qquad \forall\, i = 1, 2, .., n$$

Hyperbolic condition:

$$\prod_{h:P_h>P_i} \left(\frac{C_h}{T_h} + 1\right)\left(\frac{C_i + B_i}{T_i} + 1\right) \leq 2 \qquad \forall\, i = 1, 2, .., n$$

EDF:

Utilization factor:

$$\sum_{h:P_h>P_i} \frac{C_h}{T_h} + \frac{C_i + B_i}{T_i} \leq 1$$

Schedulability of Resource Access Protocols

RM: Response Time Analysis:

$$R_i^0 = C_i + B_i$$

$$R_i^s = C_i + B_i + \sum_{h:P_h>P_i} \left\lceil \frac{R_i^{s-1}}{T_h} \right\rceil C_h$$

Recursive iteration while $R_i^s > R_i^{s-1}$

EDF: Processor Demand Criterion: $\mathrm{U} < 1$ and $\forall\, L \in \boldsymbol{\mathcal{D}}$

$$B(L) + \sum_{k=1}^{n} \left\lfloor \frac{L + T_k - D_k}{T_k} \right\rfloor C_k \leq L$$

where $B(L) = \max\{\delta_{jh} \mid D_j > L \text{ and } D_h \leq L\}$

$$\boldsymbol{\mathcal{D}} = \{d_k \mid d_k \leq \max(D_{\max}, min(H, L^*))\}$$

$$D_{\max} = \max(D_1, D_2, \dots, D_n)$$

$$H = \mathrm{lcm}(T_1, T_2, \dots, T_n)$$

$$L^* = \frac{\sum_{j=1}^{n}(T_i - D_i)U_i}{1 - U}$$

COMPARISON OF RESOURCE ACCESS PROTOCOLS

Comparing Resource Access Protocols

PROTOCOL	PRIO	PESSIMISM	BLOCKING INSTANT	TRANS-PARENT	AVOID DEADLOCK	STACK SHARING	IMPLEMEN-TATION
NPP	Any	High	On arrival	Yes	Yes	Yes	Easy
HLP	Fixed	Medium	On arrival	No	Yes	Yes	Easy
PIP	Fixed	Low	On access	Yes	No	No	Hard
PCP	Fixed	Medium	On access	No	Yes	No	Hard
SRP	Any	Medium	On arrival	No	Yes	Yes	Easy

Summary

NPP: Disables preemption when access a CS

HLP: Priority is increased until the maximum value of the tasks sharing the resource

PIP: Priority is increased until the maximum value of the blocked task

PCP: Access the resource if priority is higher than resource ceiling. Then, priority is increased until the maximum value of the tasks blocked on the resource

SRP: Preemption is allowed whenever the priority of the task trying to preempt is greater than the running priority and its preemption level is greater than the system ceiling

Summary

MAKE CRITICAL SECTIONS AS SHORT AS POSSIBLE

Example 1: enter CS, copy global variables into local ones, exit CS
operate variables
enter CS, publish data, and exit CS

Example 2: 1 publisher Set() – multiple subscribers Get()

Example 3: Be aware of all this stuff
Code complexity is a point of fault
Code simplicity may cause critical errors
Take care of nested CS