

# Shared Resources and Blocking in Real-Time Systems

Giuseppe Lipari

<http://feanor.sssup.it/~lipari>

Scuola Superiore Sant'Anna – Pisa

December 1, 2011

## Outline

- 1 Priority inversion
- 2 Priority Inheritance Protocol
  - Nested critical sections and deadlock
  - Blocking time computation and Analysis
- 3 Priority Ceiling
- 4 Stack Resource Policy
- 5 Shared Resources on EDF
  - Priority Inheritance
  - Stack Resource Policy

## Interacting tasks

- Until now, we have considered only independent tasks
  - a task never blocks or suspends
  - it can only be suspended when it finishes its instance (job)
- However, in reality, many tasks exchange data through shared memory
- Consider as an example three periodic tasks:
  - One reads the data from the sensors and applies a filter. The results of the filter are stored in memory.
  - The second task reads the filtered data and computes some control law (updating the state and the outputs); both the state and the outputs are stored in memory;
  - finally, a third periodic task reads the outputs from memory and writes on the actuator device.
- All three tasks access data in the shared memory
- Conflicts on accessing this data in concurrency could make the data structures inconsistent.

## Resources and critical sections

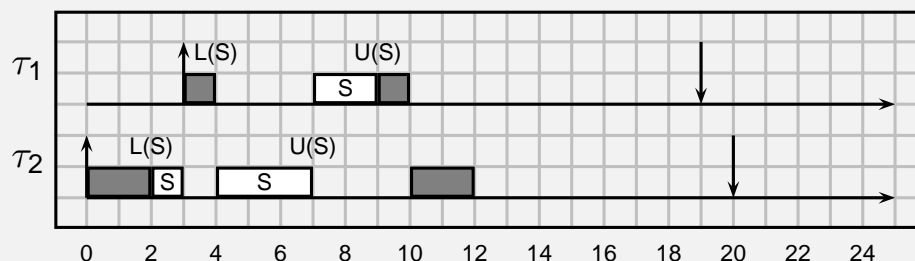
- The shared data structure is called *resource*;
- A piece of code accessing the data structure is called *critical section*;
- Two or more critical sections on the same resource must be executed in *mutual exclusion*;
- Therefore, each data structure should be *protected* by a mutual exclusion mechanism;
- In this lecture, we will study what happens when resources are protected by mutual exclusion semaphores.

## Notation

- The resource and the corresponding mutex semaphore will be denoted by symbol  $S_j$ .
- A system consists of:
  - A set of  $N$  periodic (or sporadic) tasks  $\mathcal{T} = \{\tau_1, \dots, \tau_N\}$ ;
  - A set of shared resources  $\mathcal{S} = \{S_1, \dots, S_M\}$ ;
  - We say that a task  $\tau_i$  uses resource  $S_j$  if it accesses the resource with a critical section.
  - The  $k$ -th critical of  $\tau_i$  on  $S_j$  is denoted with  $cs_{i,j}(k)$ .
  - The length of the longest critical section of  $\tau_i$  on  $S_j$  is denoted by  $\xi_{i,j}$ .

## Blocking and priority inversion

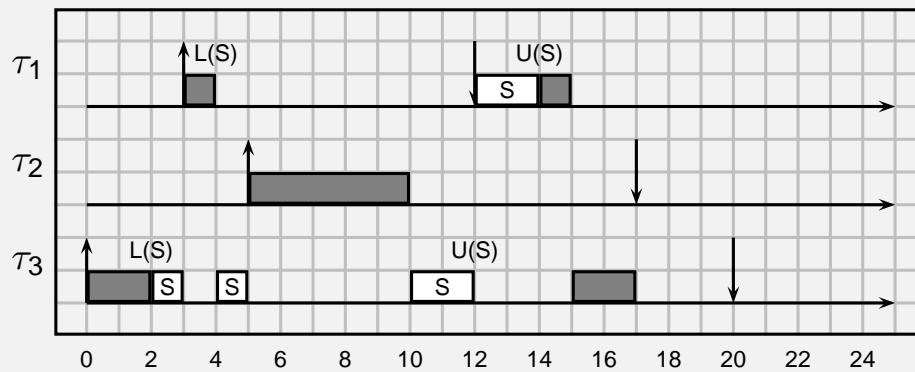
- A blocking condition happens when a high priority tasks wants to access a resource that is held by a lower priority task.
- Consider the following example, where  $p_1 > p_2$ .



- From time 4 to 7, task  $\tau_1$  is blocked by a lower priority task  $\tau_2$ ; this is a *priority inversion*.
- Priority inversion is not avoidable; in fact,  $\tau_1$  must wait for  $\tau_2$  to leave the critical section.
- However, in some cases, the priority inversion could be too large.

## Example of priority inversion

- Consider the following example, with  $p_1 > p_2 > p_3$ .



- This time the priority inversion is very large: from 4 to 12.
- The problem is that, while  $\tau_1$  is blocked,  $\tau_2$  arrives and preempt  $\tau_3$  before it can leave the critical section.
- If there are other medium priority tasks, they could preempt  $\tau_3$  as well.
- Potentially, the priority inversion could be unbounded!

## The Mars Pathfinder

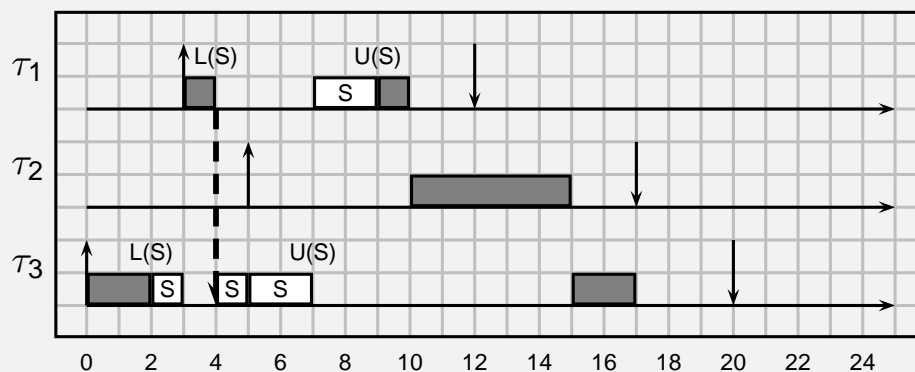
- This is not only a theoretical problem. It may happen in real cases.
- The most famous example of such problem was found during the Mars Pathfinder mission.

# The Priority Inheritance protocol

- The solution of the problem is rather simple;
  - While the low priority task blocks an higher priority task, it *inherits* the priority of the higher priority task;
  - In this way, every medium priority task cannot make preemption.

## Example

- In the previous example:



- Task  $\tau_3$  inherits the priority of  $\tau_1$
- Task  $\tau_2$  cannot preempt  $\tau_3$  ( $p_2 < p_1$ )

## Comments

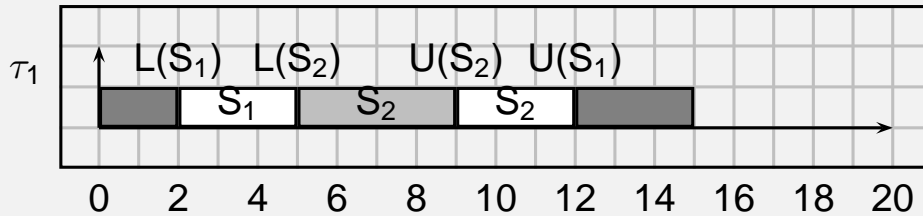
- The blocking (priority inversion) is now bounded to the length of the critical section of task  $\tau_3$
- Tasks with intermediate priority  $\tau_2$  cannot interfere with  $\tau_1$
- However,  $\tau_2$  has a blocking time, even if it does not use any resource
  - This is called *indirect blocking* (or *push-through*)
  - due to the fact that  $\tau_2$  is *in the middle between*  $\tau_1$  and  $\tau_3$  which use the same resource.
  - This blocking time must be computed and taken into account in the formula

## To be solved

- It remains to understand:
  - What is the maximum blocking time for a task
  - How we can account for blocking times in the schedulability analysis
- From now on, the maximum blocking time for a task  $\tau_i$  is denoted by  $B_i$ .

## Nested critical sections

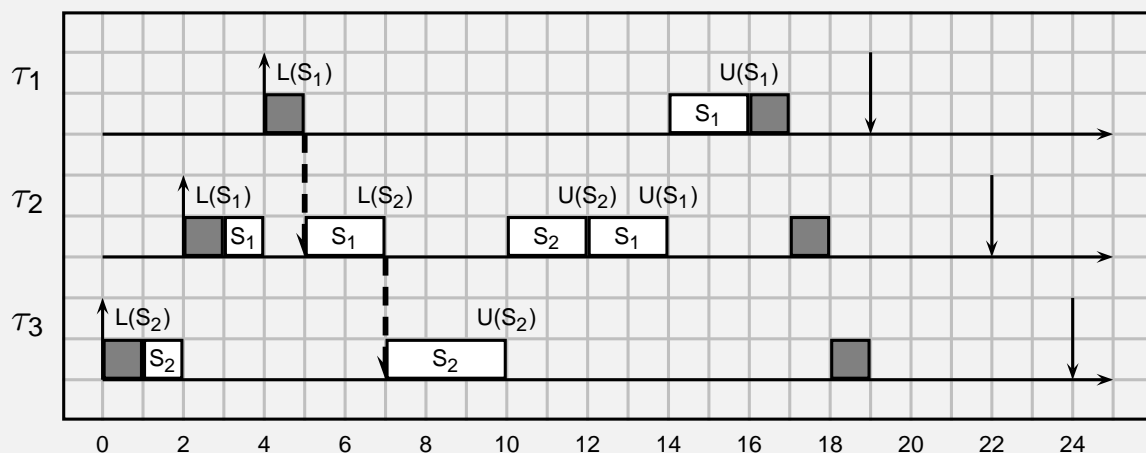
- Critical sections can be nested:
- While a task  $\tau$  is accessing a resource  $S_1$ , it can lock a resource  $S_2$ .



- When critical sections are nested, we can have *multiple inheritance*

## Multiple inheritance

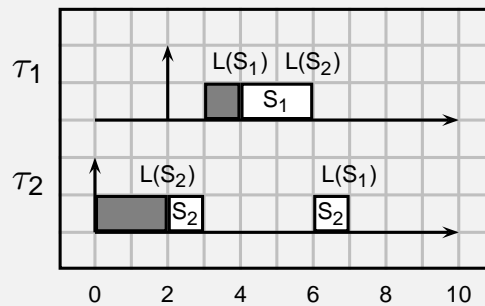
- Task  $\tau_1$  uses resource  $S_1$ ; Task  $\tau_2$  uses  $S_1$  and  $S_2$  nested inside  $S_1$ ; Task  $\tau_3$  uses only  $S_2$ .
- $p_1 > p_2 > p_3$ ;



- At time  $t = 7$  task  $\tau_3$  inherits the priority of  $\tau_2$ , which at time 5 had inherited the priority of  $\tau_1$ . Hence, the priority of  $\tau_3$  is  $p_1$ .

## Deadlock problem

- When using nested critical section, the problem of deadlock can occur; i.e. two or more tasks can be blocked waiting for each other.
- The priority inheritance protocol *does not solve* automatically the problem of deadlock, as it is possible to see in the following example.
  - Task  $\tau_1$  uses  $S_2$  inside  $S_1$ , while task  $\tau_2$  uses  $S_1$  inside  $S_2$ .



- While  $\tau_1$  is blocked on  $S_2$ , which is held by  $\tau_2$ ,  $\tau_2$  is blocked on  $S_1$  which is held by  $\tau_1$ : **deadlock!**

## Deadlock avoidance

- In the previous example, the priority inheritance protocol does not help.
- To avoid deadlock, it is possible to restrict programming freedom;
  - The problem is due to the fact that resources are accessed in a random order by  $\tau_1$  and  $\tau_2$ .
  - One possibility is to decide an order a-priori *before writing the program*.
  - Example: resources must be accessed in the order given by their index ( $S_1$  before  $S_2$  before  $S_3$ , and so on).
  - With this rule, task  $\tau_2$  is not legal because it accesses  $S_1$  inside  $S_2$ , violating the ordering.
  - If  $\tau_2$  accesses the resources in the correct order ( $S_2$  inside  $S_1$ , the deadlock is automatically avoided).



# The Priority Inheritance Protocol

- Summarising, the main rules are the following;
  - If a task  $\tau_i$  is blocked on a resource protected by a mutex semaphore  $S$ , and the resource is locked by task  $\tau_j$ , then  $\tau_j$  *inherits* the priority of  $\tau_i$ ;
  - If  $\tau_j$  is itself blocked on another semaphore by a task  $\tau_k$ , then  $\tau_k$  inherits the priority of  $\tau_i$  (*multiple inheritance*);
  - If  $\tau_k$  is blocked, the chain of blocked tasks is followed until a non-blocked task is found that inherits the priority of  $\tau_i$ .
  - When a task unlocks a semaphore, it returns to the priority it had when locking it.

## Computing the maximum blocking time

- We will compute the maximum blocking time only in the case of non nested critical sections.
  - Even if we avoid the problem of deadlock, when critical sections are nested, the computation of the blocking time becomes very complex due to multiple inheritance.
  - If critical section are not nested, multiple inheritance cannot happen, and the computation of the blocking time becomes simpler.

# Theorems

- To compute the blocking time, we must consider the following two important theorems:

## Theorem

*Under the priority inheritance protocol, a task can be blocked only once on each different semaphore.*

## Theorem

*Under the priority inheritance protocol, a task can be blocked by another lower priority task for at most the duration of one critical section.*

- A task can be blocked more than once, but only once per each resource and once by each task.

# Blocking time computation

- We must build a *resource usage table*.
  - On each row, we put a task in decreasing order of priority;
  - On each column we put a resource (the order is not important);
  - On each cell  $(i, j)$  we put  $\xi_{i,j}$ , i.e. the length of the longest critical section of task  $\tau_i$  on resource  $S_j$ , or 0 if the task does not use the resource.
- A task can be blocked only by lower priority tasks:
  - Then, for each task (row), we must consider only the rows below (tasks with lower priority).
- A task can be blocked only on resources that it uses directly, or used by higher priority tasks (*indirect blocking*);
  - For each task, we must consider only those column on which it can be blocked (used by itself or by higher priority tasks).

## Example of blocking time computation

- let's start from  $B_1$

	$S_1$	$S_2$	$S_3$	$B$
$\tau_1$	2	0	0	?
$\tau_2$	0	1	0	?
$\tau_3$	0	0	2	?
$\tau_4$	3	3	1	?
$\tau_5$	1	2	1	?

- $\tau_1$  can be blocked only on  $S_1$ .
- Therefore, we must consider only the first column, and take the maximum, which is 3.

## Example of blocking time computation

- Now  $\tau_2$ : it can be blocked on  $S_1$  (*indirect blocking*) and on  $S_2$ .

	$S_1$	$S_2$	$S_3$	$B$
$\tau_1$	2	0	0	3
$\tau_2$	0	1	0	?
$\tau_3$	0	0	2	?
$\tau_4$	3	3	1	?
$\tau_5$	1	2	1	?

- Therefore, we must consider the first 2 columns;
- Then, we must consider all cases where two distinct lower priority tasks between  $\tau_3$ ,  $\tau_4$  and  $\tau_5$  access  $S_1$  and  $S_2$
- The possibilities are:
  - $\tau_4$  on  $S_1$  and  $\tau_5$  on  $S_2$ :  $\rightarrow 5$ ;
  - $\tau_4$  on  $S_2$  and  $\tau_5$  on  $S_1$ :  $\rightarrow 4$ ;
- The maximum is  $B_2 = 5$ .

## Example of blocking time computation

- $\tau_3$  can be blocked on all 3 resources

	$S_1$	$S_2$	$S_3$	$B$
$\tau_1$	2	0	0	3
$\tau_2$	0	1	0	5
$\tau_3$	0	0	2	?
$\tau_4$	3	3	1	?
$\tau_5$	1	2	1	?

- The possibilities are:
  - $\tau_4$  on  $S_1$  and  $\tau_5$  on  $S_2$ :  $\rightarrow 5$ ;
  - $\tau_4$  on  $S_2$  and  $\tau_5$  on  $S_1$  or  $S_3$ :  $\rightarrow 4$ ;
  - $\tau_4$  on  $S_3$  and  $\tau_5$  on  $S_1$ :  $\rightarrow 2$ ;
  - $\tau_4$  on  $S_3$  and  $\tau_5$  on  $S_2$  or  $S_3$ :  $\rightarrow 3$ ;
- The maximum is  $B_3 = 5$ .

## Example of blocking time computation

- Now:

	$S_1$	$S_2$	$S_3$	$B$
$\tau_1$	2	0	0	3
$\tau_2$	0	1	0	5
$\tau_3$	0	0	2	5
$\tau_4$	3	3	1	?
$\tau_5$	1	2	1	?

- $\tau_4$  can be blocked on all 3 resources, but only by  $\tau_5$ .
- The maximum is  $B_4 = 2$ .
- $\tau_5$  cannot be blocked by any other task (because it is the lower priority task!);  $B_5 = 0$ ;

## Example: Final result

- the final result is:

	$S_1$	$S_2$	$S_3$	$B$
$\tau_1$	2	0	0	3
$\tau_2$	0	1	0	5
$\tau_3$	0	0	2	5
$\tau_4$	3	3	1	2
$\tau_5$	1	2	1	0

## Scheduling analysis

- In case of relative deadlines equal to periods, we have:

$$\forall i = 1, \dots, N \quad \sum_{j=1}^i \frac{C_j}{T_j} + \frac{B_i}{T_i} \leq U_{\text{lub}}$$

## Example

- In the previous example:

	$C$	$T$	$U$	$B$
$\tau_1$	4	16	0.25	3
$\tau_2$	3	24	0.125	5
$\tau_3$	4	32	0.125	5
$\tau_4$	5	40	0.125	2
$\tau_5$	4	50	0.08	0

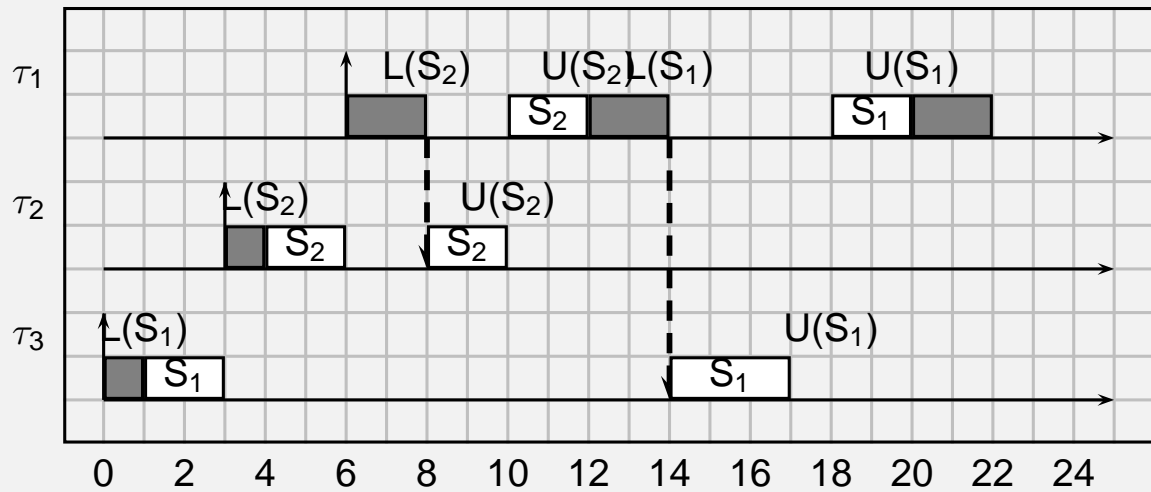
- $U_1 + \frac{B_1}{T_1} = 0.25 + 0.1875 = 0.4375 < 0.743$
- $U_1 + U_2 + \frac{B_2}{T_2} = 0.375 + 0.208 = 0.583 < 0.743$
- $U_1 + U_2 + U_3 + \frac{B_3}{T_3} = 0.5 + 0.156 = 0.656 < 0.743$
- $U_1 + U_2 + U_3 + U_4 + \frac{B_4}{T_4} = 0.625 + 0.05 = 0.675 < 0.743$
- $U_1 + U_2 + U_3 + U_4 + U_5 \frac{B_5}{T_5} = 0.705 + 0 < 0.743$

## Problems of Priority inheritance

- Multi blockings
  - A task can be blocked more than once on different semaphores
- Multiple inheritance
  - when considering nested resources, the priority can be inherited multiple times
- Deadlock
  - In case of nested resources, there can be a deadlock

## Multiple blocking example

- example:



- task  $\tau_1$  is blocked twice on two different resources

## Possible solution: ceilings

- It is possible to avoid this situation by doing an off-line analysis
- Define the concept of resource ceiling
- Anticipate the blocking: a task cannot lock a resource if **it can potentially block** another higher priority task later.

### Definition

The ceiling of a resource is the priority of the highest priority task that can access it

$$\text{ceil}(S_k) = \max_i \{p_i \mid \tau_i \text{ uses } S_k\}$$

# The priority ceiling protocol

- Basic rules
- When a task  $\tau_i$  tries to lock a semaphore  $S_k$ , the operation is permitted only if the following two conditions hold:

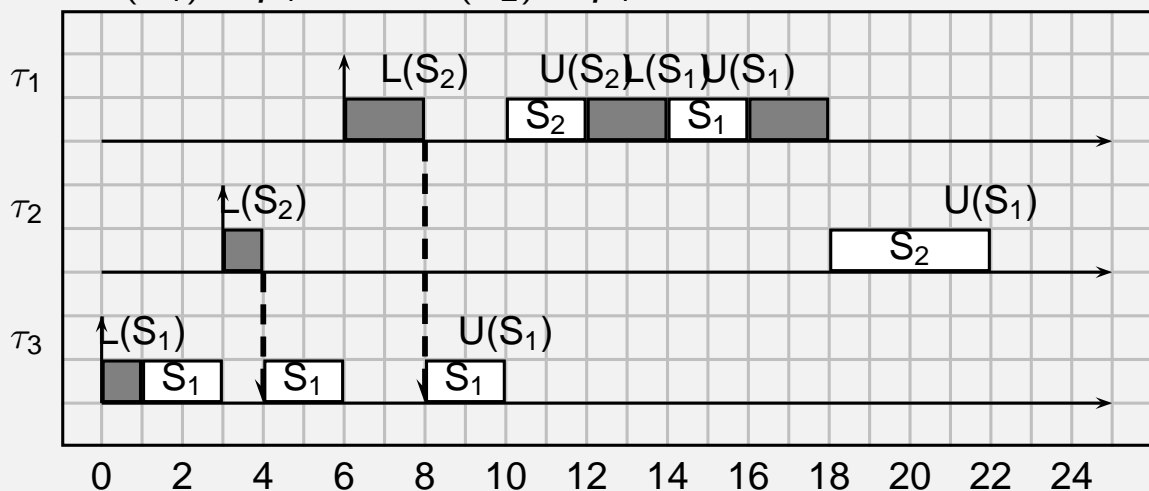
- 1  $S_k$  is free and
- 2  $p_i > \text{maxceil}_i$  where:

$$\text{maxceil}_i = \max \{ \text{ceil}(S_j) \mid S_j \text{ is locked by a task different from } \tau_i \}$$

- Otherwise, the task is blocked, and the *blocking task* inherits the priority
- The blocking task is the one that holds the lock on the semaphore corresponding to the maximum ceiling  $\text{maxceil}_i$ .

## Example:

- $\text{ceil}(S_1) = p_1$  and  $\text{ceil}(S_2) = p_1$

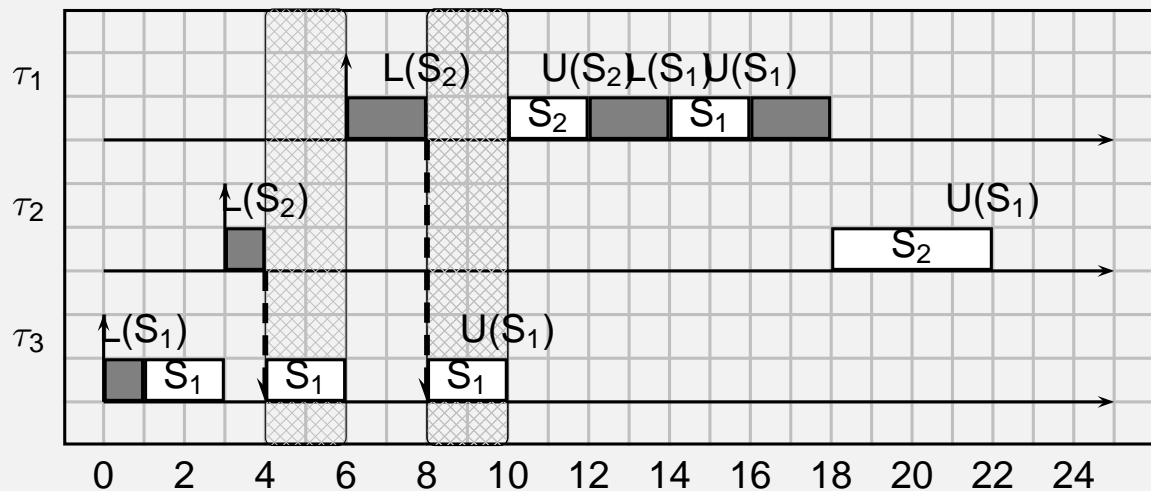


- task  $\tau_3$  acquires the lock
- task  $\tau_2$  is blocked because  $p_2 < \text{maxceil} = p_1$
- task  $\tau_3$  inherits  $\tau_2$ 's priority
- task  $\tau_1$  is blocked for the same reason
- task  $\tau_3$  inherits  $\tau_1$ 's priority
- task  $\tau_3$  returns to its original priority, since it is not blocking anyone



# Blocking

- In the previous example:



- Blocking time for  $\tau_1$ : 2
- Blocking time for  $\tau_2$ : 4
- No multiple blockings!

## Properties of PCP

### Theorem

*A task can be blocked at most once by any resource or task.*

### Theorem

*The Priority Ceiling Protocol prevents deadlock*

- Therefore, we can nest critical sections safely

### Corollary

*The maximum blocking time for a task is at most the length of one critical section*

- It follows that in the resource table, we have to consider only one cell

## Example of blocking time computation – PCP

- let's start from  $B_1$

	$S_1$	$S_2$	$S_3$	$B$
$\tau_1$	2	0	0	?
$\tau_2$	0	1	0	?
$\tau_3$	0	0	2	?
$\tau_4$	3	3	1	?
$\tau_5$	1	2	1	?

- $\tau_1$  can be blocked only on  $S_1$ .
- Therefore, we must consider only the first column, and take the maximum, which is 3.

## Example of blocking time computation – PCP

- Now  $\tau_2$ : it can be blocked on  $S_1$  (*indirect blocking*) and on  $S_2$ .

	$S_1$	$S_2$	$S_3$	$B$
$\tau_1$	2	0	0	3
$\tau_2$	0	1	0	?
$\tau_3$	0	0	2	?
$\tau_4$	3	3	1	?
$\tau_5$	1	2	1	?

- Therefore, we must consider the first 2 columns;
- The maximum is  $B_2 = 3$ .

## Example of blocking time computation – PCP

- $\tau_3$  can be blocked on all 3 resources

	$S_1$	$S_2$	$S_3$	$B$
$\tau_1$	2	0	0	3
$\tau_2$	0	1	0	3
$\tau_3$	0	0	2	?
$\tau_4$	3	3	1	?
$\tau_5$	1	2	1	?

- The maximum is  $B_3 = 3$ .

## Example of blocking time computation – PCP

- Now :

	$S_1$	$S_2$	$S_3$	$B$
$\tau_1$	2	0	0	3
$\tau_2$	0	1	0	3
$\tau_3$	0	0	2	3
$\tau_4$	3	3	1	?
$\tau_5$	1	2	1	?

- $\tau_4$  can be blocked on all 3 resources,
- The maximum is  $B_4 = 2$ .
- $\tau_5$  cannot be blocked by any other task (because it is the lower priority task!);  $B_5 = 0$ ;

## Example: Final result – PCP

- Final result:

	$S_1$	$S_2$	$S_3$	$B$
$\tau_1$	2	0	0	3
$\tau_2$	0	1	0	3
$\tau_3$	0	0	2	3
$\tau_4$	3	3	1	2
$\tau_5$	1	2	1	0

## PCP – problems

- The PCP has some disadvantages
- The implementation is very complex, even more than PI
  - Very little known implementations,
  - difficult to prove correctness of implementation
- The PCP causes still many context switches
- we need something simpler to be implemented!

# Stack Resource Policy

- This protocol is also known with the name of Immediate Priority Ceiling Protocol (IPCP).  
The basic ideas are the following:
  - We anticipate the blocking even more
  - the task cannot even start executing if it is not guaranteed to take all resources
- Properties:
  - Very simple implementation
  - A task blocks at most once before starting execution
  - The execution order is like a “stack”.

## Preemption levels and rules

- The **preemption level**  $\pi_i$  of a task is a generalization of the concept of priority for preemptive scheduling;
- In Fixed Priority, the preemption level of a task is defined as its priority:  $\pi_i = p_i$ .
- In EDF the preemption level will be defined as the inverse of the relative deadline of a task.

### Definition

The ceiling of a resource is the preemption level of the task with the highest preemption level among those that can access the resource

$$\text{ceil}(S_k) = \max_i \{ \pi_i | \tau_i \text{ uses } S_k \}$$

### Definition

The system ceiling at any instant of time is the maximum ceiling among all locked resources

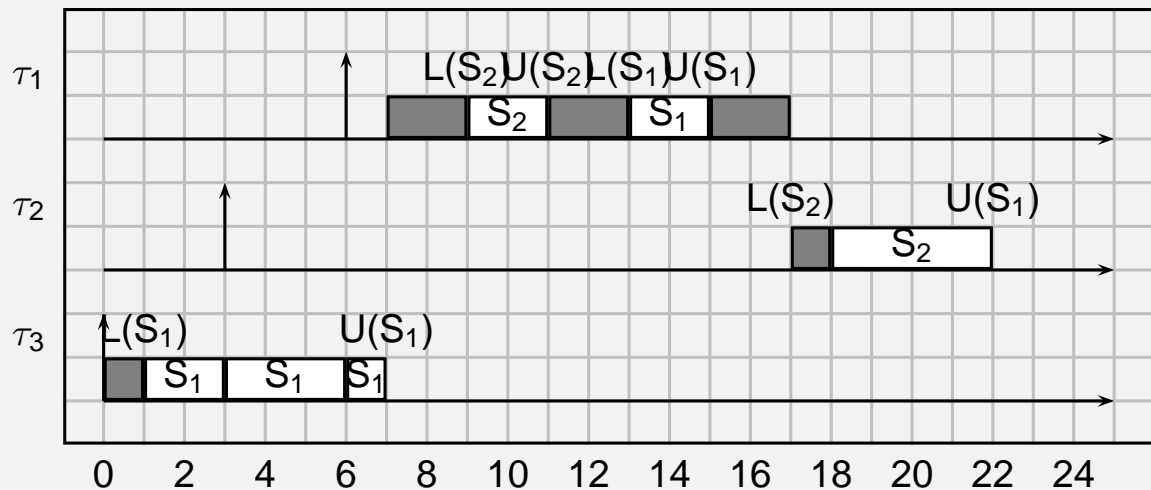
$$\Pi_s(t) = \max \{ \text{ceil}(S) | S \text{ is locked at } t \}$$

# SRP rule

- The protocol rule is:
- A task that arrives at time  $t$  can start executing only if:
  - 1 it is the highest priority task
  - 2 its preemption level is greater than current system ceiling:  
 $\pi_i > \Pi_s(t)$

## Example

- Example



- Task  $\tau_3$  raises the sys ceiling to  $p_1$
- Task  $\tau_2$  cannot start because  $p_2 < \Pi_s = p_1$
- Task  $\tau_1$  cannot start because  $p_1 = \Pi_s = p_1$
- When task  $\tau_3$  unlocks, the sys ceiling goes down, and all other tasks can start executing

# Properties

The same properties of PCP hold, in particular:

## Theorem

*A task can be blocked at most once by any resource or task.*

## Theorem

*The Stack Resource Policy prevents deadlock*

Therefore, we can nest critical sections safely

## Corollary

*The maximum blocking time for a task is at most the length of one critical section*

Therefore, the blocking time is the same as PCP.

It can be proven that this is the minimal possible blocking time

# SRP vs. PCP

- SRP reduces the number of preemptions
- SRP is very easy to be implemented
  - No need to do inheritance
  - No need to block tasks in semaphore queues
  - It makes it possible for all tasks to share the same stack

## Non preemptive scheduling

- Using SRP is equivalent to selectively disable preemption for a limited amount of time
- We can disable preemption only for some group of tasks
- The SRP is a generalization of the preemption-threshold mechanism

## Synchronization protocols with EDF

- Both the Priority inheritance Protocol and the Stack Resource Policy can be used under EDF without any modification.
- Let's first consider PI.
  - When a higher priority *job* is blocked by a lower priority job on a shared mutex semaphore, then the lower priority job *inherits* the priority of the blocked job.
  - In EDF, the priority of a job is inversely proportional to its absolute deadline.
  - Here, you should substitute *higher priority job* with **job with an early deadline** and *inherits the priority* with **inherits the absolute deadline**.



## Preemption levels

- To compute the blocking time, we must first order the tasks based on their *preemption levels*.

### Definition

Every task  $\tau_i$  is assigned a preemption level  $\pi_i$  such that it can preempt a task  $\tau_j$  if and only if  $\pi_i > \pi_j$ .

- In fixed priority, the preemption level is the same as the priority.
- In EDF, the preemption level is defined as  $\pi_i = \frac{1}{D_i}$ .

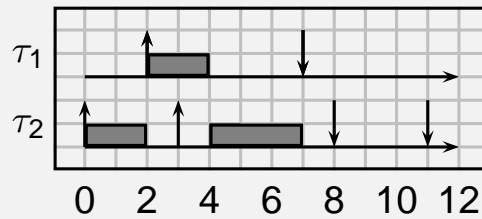
## Preemption Levels - II

- If  $\tau_i$  can preempt  $\tau_j$ , then the following two conditions must hold:
- $\tau_i$  arrives after  $\tau_j$  has started to execute and hence  $a_i > a_j$ ,
- the absolute deadline of  $\tau_i$  is shorter than the absolute deadline of  $\tau_j$  ( $d_i \leq d_j$ ).
- It follows that

$$\begin{aligned}d_i &= a_i + D_i \leq d_j = a_j + D_j \Rightarrow \\D_i - D_j &\leq a_j - a_i < 0 \Rightarrow \\D_i &< D_j \Rightarrow \\\pi_i &> \pi_j\end{aligned}$$

## Preemption levels

- With a graphical example:



- Notice that  $\pi_1 > \pi_2$ ;
- In this case,  $\tau_1$  preempts  $\tau_2$ .
- $\tau_2$  cannot preempt  $\tau_1$  (because its relative deadline is greater than  $\tau_1$ ).

## Computing the blocking time

- To compute the blocking time for EDF + PI, we use the same algorithms as for FP + PI. In particular, the fundamental theorem for PI is still valid:

### Theorem

*Each task can be blocked only once per each resource, and only for the length of one critical section per each task.*

## Computing the blocking time - II

- In case on non-nested critical sections, build a *resource usage table*
  - At each row put a task, ordered by decreasing preemption levels
  - At each column, put a resource
  - In each cell, put the worst case duration  $\xi_{ij}$  of any critical section of task  $\tau_i$  on resource  $S_j$
- The algorithm for the blocking time for task  $\tau_i$  is the same:
  - Select the rows below the  $i$ -th;
  - we must consider only those column on which it can be blocked (used by itself or by higher priority tasks)
  - Select the maximum sum of the  $\xi_{k,j}$  with the limitation of at most one  $\xi_{k,j}$  for each  $k$  and for each  $j$ .

## Schedulability formula

- In case of relative deadlines equal to periods, we have:

$$\forall i = 1, \dots, N \quad \sum_{j=1}^i \frac{C_j}{T_j} + \frac{B_i}{T_i} \leq 1$$

- In case of relative deadlines less than the periods:

$$\begin{aligned} \forall i = 1, \dots, N \quad \forall L < L^* \\ \sum_{j=1}^N \left( \left\lfloor \frac{L - D_j}{T_j} \right\rfloor + 1 \right) C_j + B_i \leq L \\ L^* = \frac{U}{1 - U} \max_i (T_i - D_i) \end{aligned}$$

## Complete example

- Here we analyze a complete example, from the parameters of the tasks, and from the resource usage table, we compute the  $B_i$ s, and test schedulability.

	$C_i$	$T_i$	$U_i$	$R_1$	$R_2$	$B_i$
$\tau_1$	2	10	.2	1	0	?
$\tau_2$	5	15	.33	2	1	?
$\tau_3$	4	20	.2	0	2	?
$\tau_4$	9	45	.2	3	4	?

## Complete example: blocking times

- Blocking time for  $\tau_1$ :

	$C_i$	$T_i$	$U_i$	$R_1$	$R_2$	$B_i$
$\tau_1$	2	10	.2	1	0	3
$\tau_2$	5	15	.33	2	1	?
$\tau_3$	4	20	.2	0	2	?
$\tau_4$	9	45	.2	3	4	?

## Complete example: blocking times

- Blocking time for  $\tau_2$ :

	$C_i$	$T_i$	$U_i$	$R_1$	$R_2$	$B_i$
$\tau_1$	2	10	.2	1	0	3
$\tau_2$	5	15	.33	2	1	5
$\tau_3$	4	20	.2	0	2	?
$\tau_4$	9	45	.2	3	4	?

## Complete example: blocking times

- Blocking time for  $\tau_3$ :

	$C_i$	$T_i$	$U_i$	$R_1$	$R_2$	$B_i$
$\tau_1$	2	10	.2	1	0	3
$\tau_2$	5	15	.33	2	1	5
$\tau_3$	4	20	.2	0	2	4
$\tau_4$	9	45	.2	3	4	?

## Complete example: blocking times

- Blocking time for  $\tau_4$ :

	$C_i$	$T_i$	$U_i$	$R_1$	$R_2$	$B_i$
$\tau_1$	2	10	.2	1	0	3
$\tau_2$	5	15	.33	2	1	5
$\tau_3$	4	20	.2	0	2	4
$\tau_4$	9	45	.2	3	4	0

## Complete Example: schedulability test

- General formula:

$$\forall i = 1, \dots, 4 \quad \sum_{j=1}^i \frac{C_j}{T_j} + \frac{B_i}{T_i} \leq 1$$

- Task  $\tau_1$ :

$$\frac{C_1}{T_1} + \frac{B_1}{T_1} = .2 + .3 = .5 \leq 1$$

- Task  $\tau_2$ :

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{B_2}{T_2} = .5333 + .3333 = .8666 \leq 1$$

- Task  $\tau_3$ :

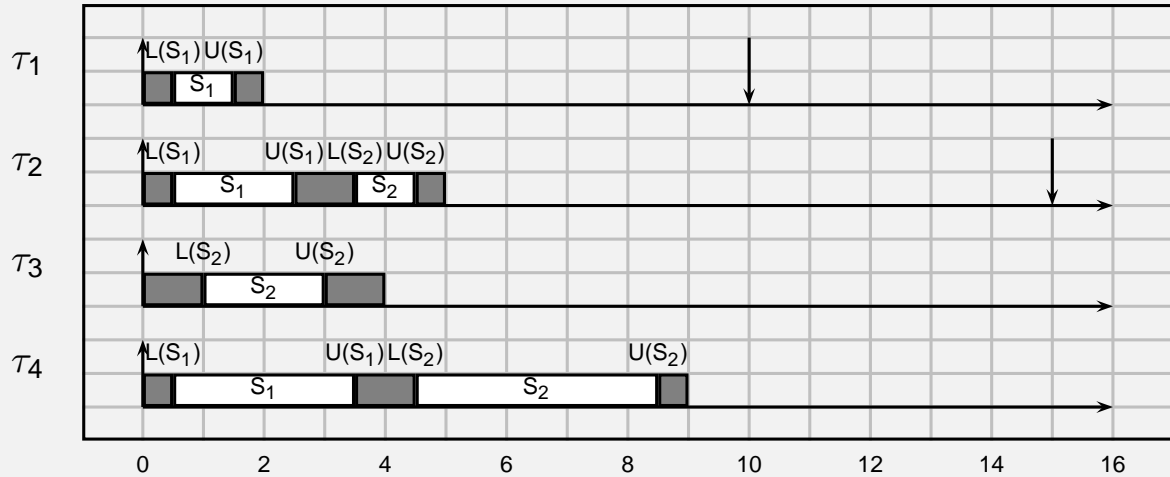
$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_3}{T_3} + \frac{B_3}{T_3} = .2 + .333 + .2 + .2 = 0.9333 \leq 1$$

- Task  $\tau_4$ :

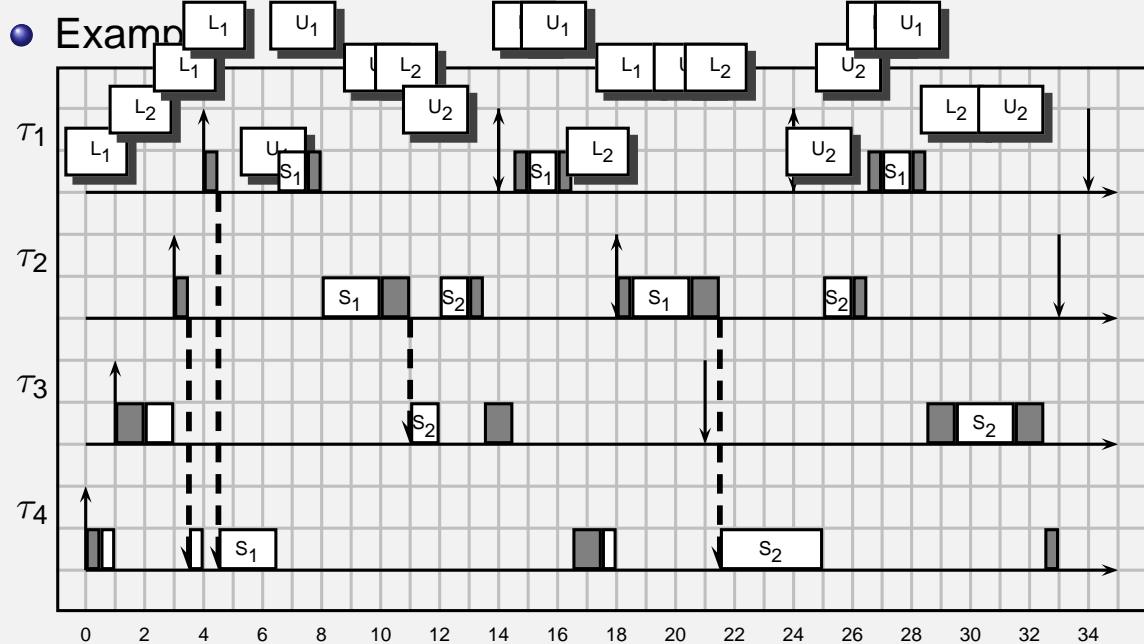
$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_3}{T_3} + \frac{C_4}{T_4} + \frac{B_4}{T_4} = .2 + .3333 + .2 + .2 + 0 = .9333 \leq 1$$

## Complete example: scheduling

- Now we do an example of possible schedule.
- We assume that the task access the resources as follows:



## Complete example: schedule



- In the graph,  $L_1 = \text{Lock}(S_1)$ ,  $U_1 = \text{Unlock}(S_1)$ ,  $L_2 = \text{Lock}(S_2)$ ,  $U_2 = \text{Unlock}(S_2)$ .
- The tasks start with an offset, because in the example we want to highlight the blocking times at the beginning.

## Stack Resource Policy

- Once we have defined the preemption levels, it is easy to extend the stack resource policy to EDF.
- The main rule is the following:
  - The *ceiling* of a resource is defined as the highest preemption level among the ones of all tasks that access it;
  - At each instant, the *system ceiling* is the highest among the ceilings of the locked resources;
  - A task is not allowed to start executing until its deadline is the shortest one and its preemption level is strictly greater than the system ceiling;

## Complete Example

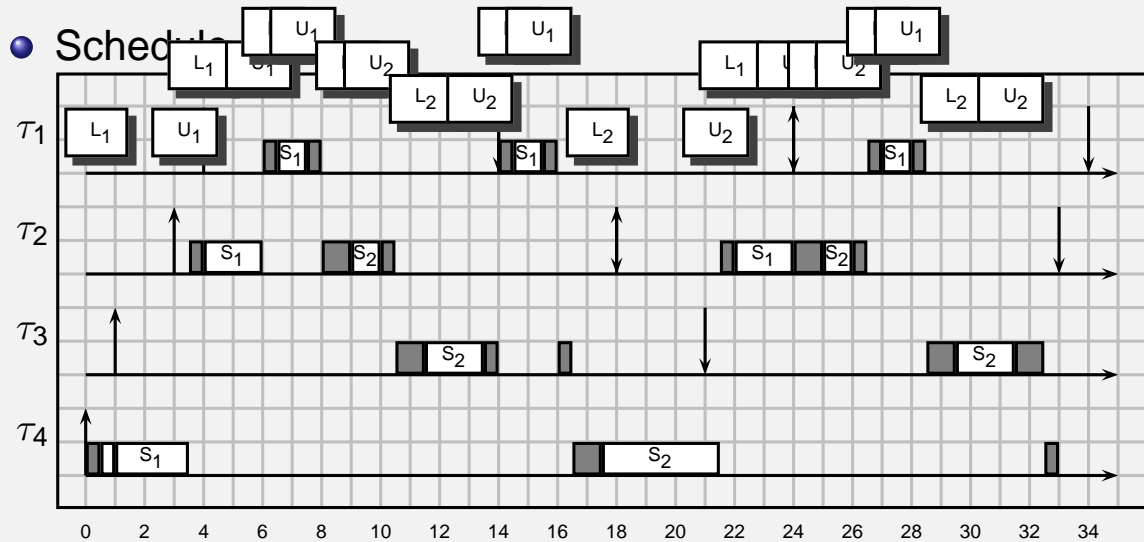
- Now we analyze the previous example, assuming EDF+SRP.

	$C_i$	$T_i$	$U_i$	$R_1$	$R_2$	$B_i$
$\tau_1$	2	10	.2	1	0	?
$\tau_2$	5	15	.33	2	1	?
$\tau_3$	4	20	.2	0	2	?
$\tau_4$	9	45	.2	3	4	?

- Let us first assign the preemption levels.
  - The actual value of the preemption levels is not important, as long as they are assigned in the right order.
  - To make calculations easy, we set  $\pi_1 = 4$ ,  $\pi_2 = 3$ ,  $\pi_3 = 2$ ,  $\pi_4 = 1$ .
- Then the resource ceilings:
  - $\text{ceil}(R_1) = \pi_1 = 4$ ,  $\text{ceil}(R_2) = \pi_2 = 3$ .



## Schedule



- At this point, the system ceiling is raised to  $\pi_1$  (the ceiling of  $R_1$ ). Task  $\tau_3$  cannot start executing, because  $\pi_3 < \pi_1$ . Same for  $\tau_2$ .
- The system ceiling goes back to 0. Now  $\tau_2$  can start.
- in this example, we assume that  $\tau_2$  locks  $R_1$  just before  $\tau_1$  arrives. Then, sys ceil =  $\pi_1$  and  $\tau_1$  cannot preempt.

## Blocking time computation

- The computation of the blocking time is the same as in the case of FP + SRP;
- The only difference is that, when the resource access table is built, tasks are ordered by decreasing preemption level, instead than by priority.
- In the previous example:

	$C_i$	$T_i$	$U_i$	$R_1$	$R_2$	$B_i$
$\tau_1$	2	10	.2	1	0	3
$\tau_2$	5	15	.33	2	1	4
$\tau_3$	4	20	.2	0	2	4
$\tau_4$	9	45	.2	3	4	0

- Notice that, since the blocking times in this case are less than in the case of Priority Inheritance, then the system is schedulable. As an exercise, check that the schedulability condition holds.