Exclusive Access to Shared Resources

Real-Time Operative Systems Course

Paulo Pedreiras, DETI/UA/IT

October 31, 2022

Last lecture

- Online scheduling with dynamic priorities
  - Earliest Deadline First scheduling
  - Analysis: CPU utilization bound and CPU Load
- Optimality and comparison with RM and SCS
  - Schedulability level, number of preemptions, jitter and response time
- Other dynamic priority criteria
  - Least Slack First, First Come First Served

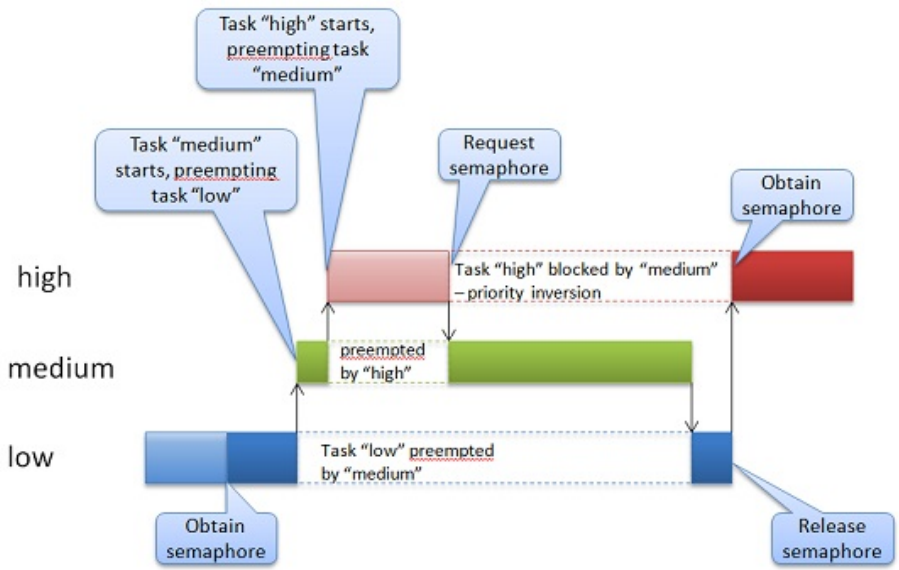
Agenda for today

- Priority inversion as a consequence of blocking
- Basic techniques to enforce exclusive access to shared resources:
  - Priority Inheritance Protocol – PIP
  - Priority Ceiling Protocol – PCP
  - Stack Resource Protocol- SRP

The priority inversion problem in not "academic"!

What really happened to the software on the Mars Pathfinder spacecraft?

- *(July 4th 1997)... the Mars Pathfinder landed to a media fanfare and began to transmit data back to Earth. Days later and the flow of information and images was interrupted by a series of total systems resets. ...*



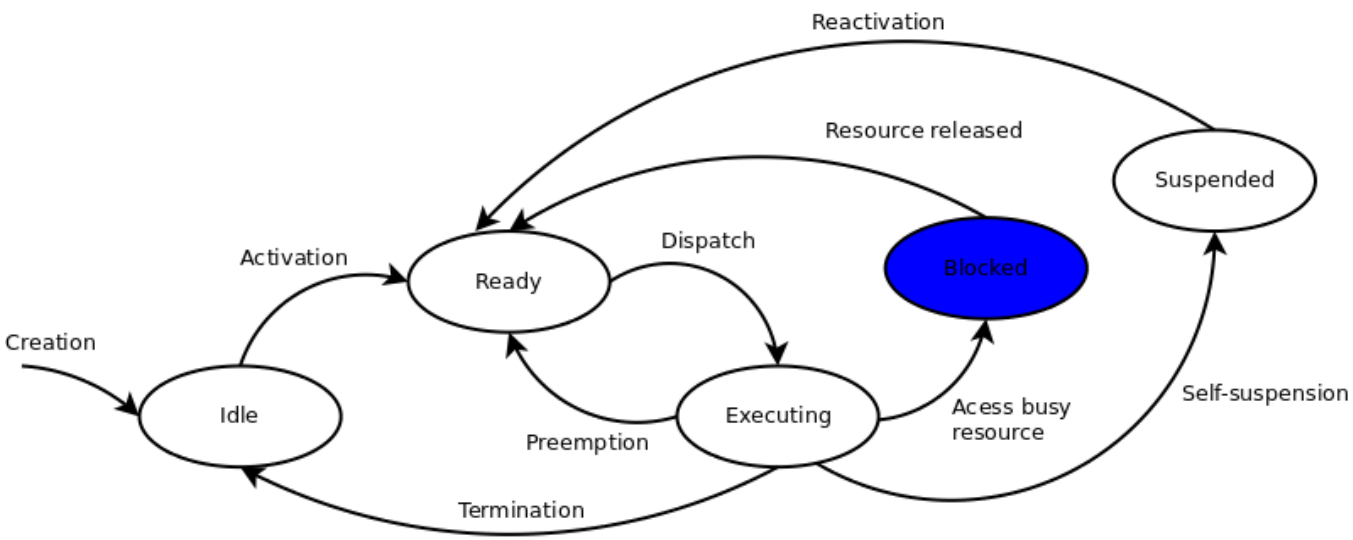
Source:

https://www.rapitasystems.com/blog/what-really-happened-to-the-software-on-the-mars-pathfinder-spacecraft

Shared resources with exclusive access

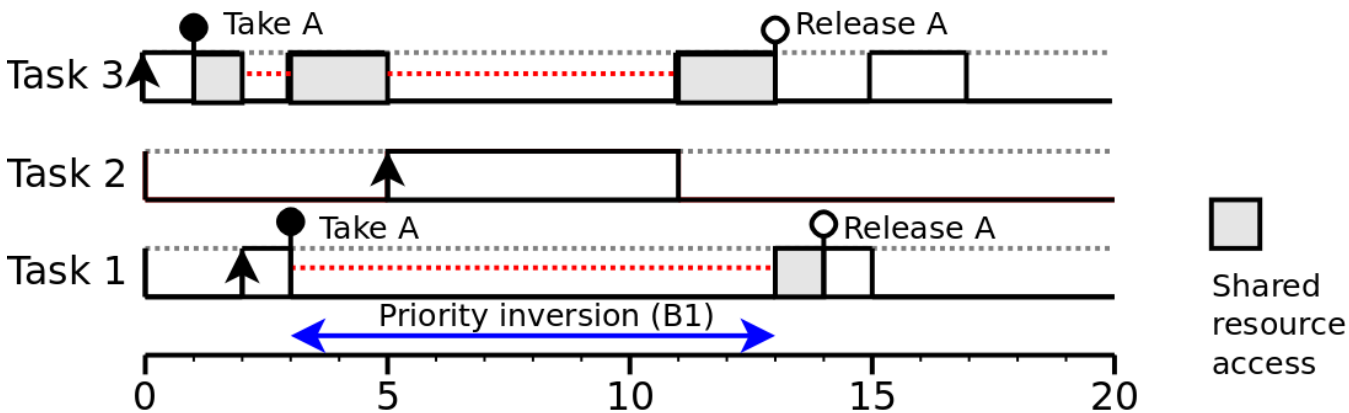
Tasks: the Blocked state

- When a running task tries to access a shared resource (e.g. a buffer, a communication port) that is **already taken** (i.e. in use) by another task, it gets **blocked** . When the resource becomes free, the blocked task becomes again ready for execution. To handle this scenario the state diagram is updated as follows:



The priority inversion phenomenon

- On a real-time system with preemption and independent tasks, the highest priority ready task is always the one in execution
- However, when tasks share resources with exclusive access, the case is different. An higher priority task may be **blocked** by another (**lower priority**) task, whenever this latter one owns a resource needed by the first one. In such scenario it is said that the higher priority task is blocked.
- When the blocking task (and eventually other tasks with intermediate priority) execute, there is a **priority inversion** .



The priority inversion phenomenon

- The priority inversion is an unavoidable phenomenon in the presence of shared resources with exclusive access.
- However, in real-time systems, it is of utmost importance to **bound and quantify** its worst-case impact, to allow reasoning about the schedulability of the task set.
- Therefore, the techniques used to guarantee the exclusive access to the resources (synchronization primitives) must **restrict the duration of the priority inversion and be analyzable** , i.e., allow the quantification of the maximum blocking time that each task may experience in any shared resource.

Techniques to allow exclusive access

Possible synchronization approaches

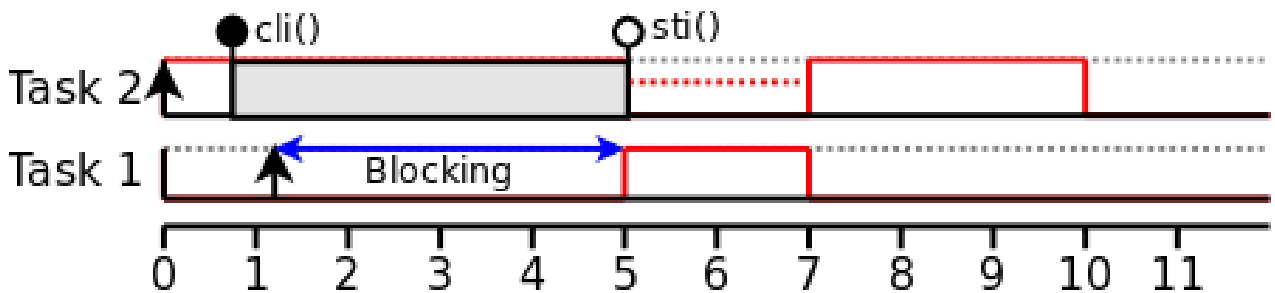
- **Disable Interrupts**
  - *disable*()/*enable*() or *cli*()/*sti*()
- **Inhibit preemption**
  - *no_preemp*()/*preempt*()
- **Locks or atomic flags**
  - *lock*()/*unlock*()
- **Use of semaphores**
  - Counter + task list
  - *P*()/*V*(), *wait*()/*signal*(), *take*()/*give*(), ...

Let's take a look at them ...

Techniques to allow exclusive access

Interrupt inhibit

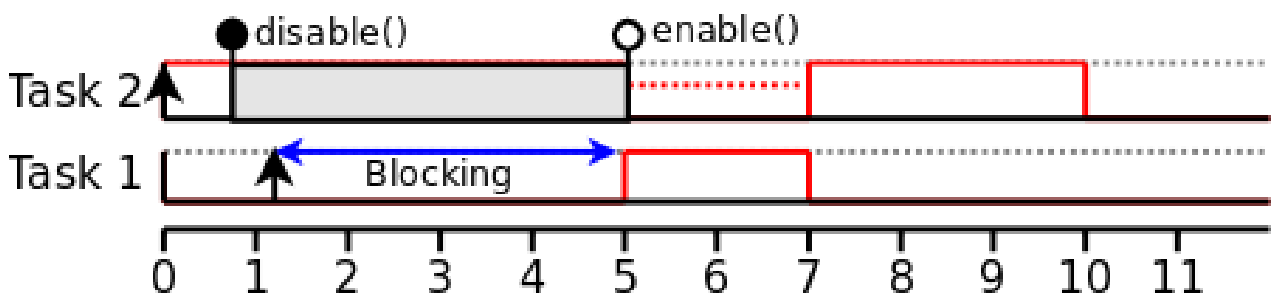
- All other system activities are blocked, not just other tasks, but also interrupt service routines, **including the system tick handler** .
- Very easy to implement but should only be used with very short critical regions (e.g. access to a few HW registers, read-modify-write of a variable)
- Each task can only be blocked once and for the maximum duration of the critical region of lower priority tasks (or smaller relative deadline for EDF), even if they don't use any shared resource!!



Techniques to allow exclusive access

Preemption inhibiting

- All other tasks are blocked. However, contrarily to disabling the interrupts, in this case the **interrupt service routines**, including the system tick, are **not blocked** !
- Very easy to implement but not efficient, as it causes unnecessary blocking.
- Each task can only be blocked once and for the maximum duration of the critical region of lower priority tasks (or smaller relative deadline for EDF), even if these don't use any shared resource!!



Techniques to allow exclusive access

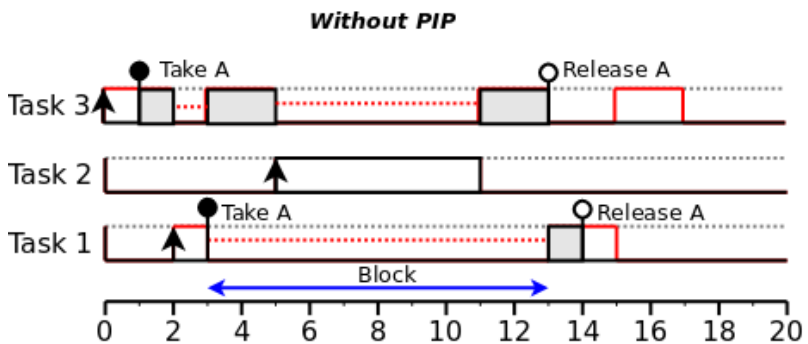
Semaphores

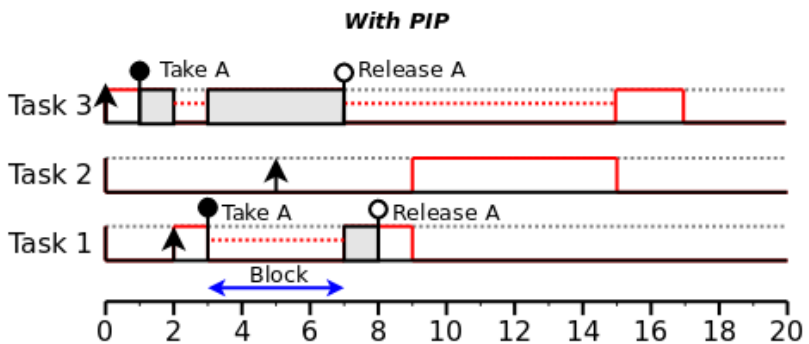
- More **complex and costly** to implement
- In general much **more efficient** resource management
- However, the blocking duration depends on the specific protocol used to manage the semaphores
- These protocols should prevent:
  - Indeterminate blocking
  - Chain blocking
  - Deadlocks

Let's see a few protocols commonly used in RTS

PIP – Priority Inheritance Protocol

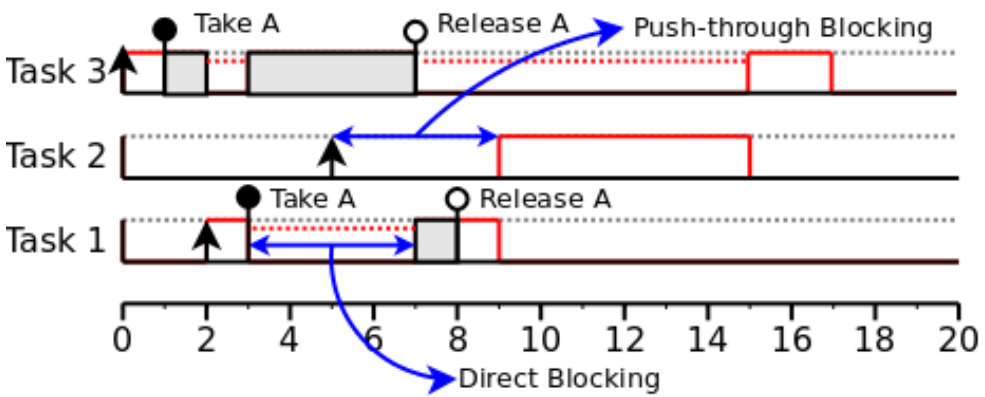
- The blocking task (lower priority) **temporarily inherits** the priority of the blocked task (the one with higher priority).
- Limits the blocking duration, **preventing the execution of intermediate priority tasks** while the blocking tasks owns the critical region. The priority of the blocking task returns to its nominal value when leaving the critical region.





PIP – Priority Inheritance Protocol

- To bound the blocking time (B) it is important to note that a task can be blocked by any lower priority task that:
  - Shares a resource with it - **Direct blocking** , or
  - Can block a task with higher priority - **Push-through** or **Indirect blocking**
- Note also that in the absence of chained accesses:
  - Each task can block any other task just once
  - Each task can block only once in each resource



τ	S_1	S_2	S_3	B_i
τ_1	1	2	0	?
τ_2	0	9	3	?
τ_3	8	7	0	?
τ_4	6	5	4	?

PIP – Priority Inheritance Protocol

Worst-case blocking times

τ	S_1	S_2	S_3
τ_1	1	2	0
τ_2	0	9	3
τ_3	8	7	0
τ_4	6	5	4

$\Rightarrow$

τ	C_i	T_i	B_i
τ_1	5	30	17
τ_2	15	60	13
τ_3	20	80	6
τ_4	20	100	0

Schedulability analysis with PiP

Utilization test, for Rate Monotonic

$$\forall_{1 \le i \le N} \sum_{h:P_h > P_i} \left(\frac{C_h}{T_h}\right) + \frac{C_i + B_i}{T_i} \le i \cdot (2^{\frac{1}{i}} - 1)$$

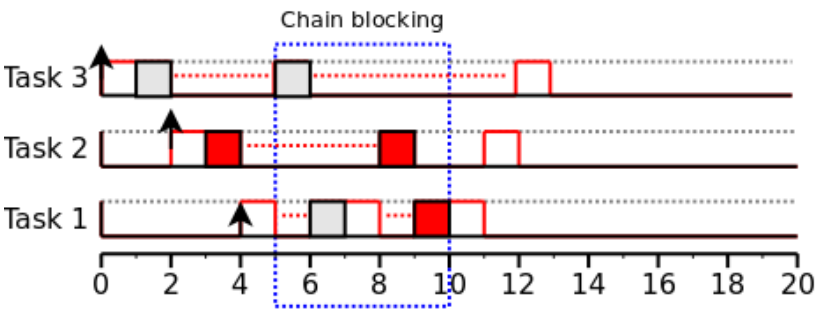
Response Time Analysis (Fixed Priority)

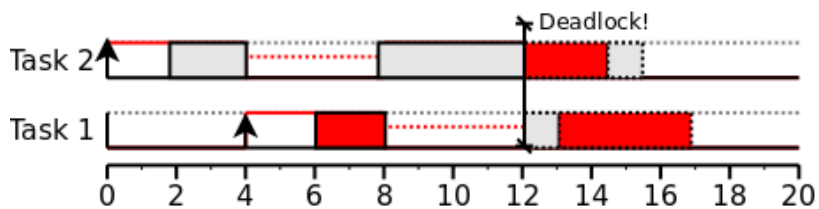
$$R_i(0) = C_i + B_i$$
$$R_i(m) = C_i + B_i + \sum_{h:P_h > Pi} \left\lceil \frac{R_i(m-1)}{T_h} \right\rceil C_h$$

PIP – Priority Inheritance Protocol

PiP evaluation:

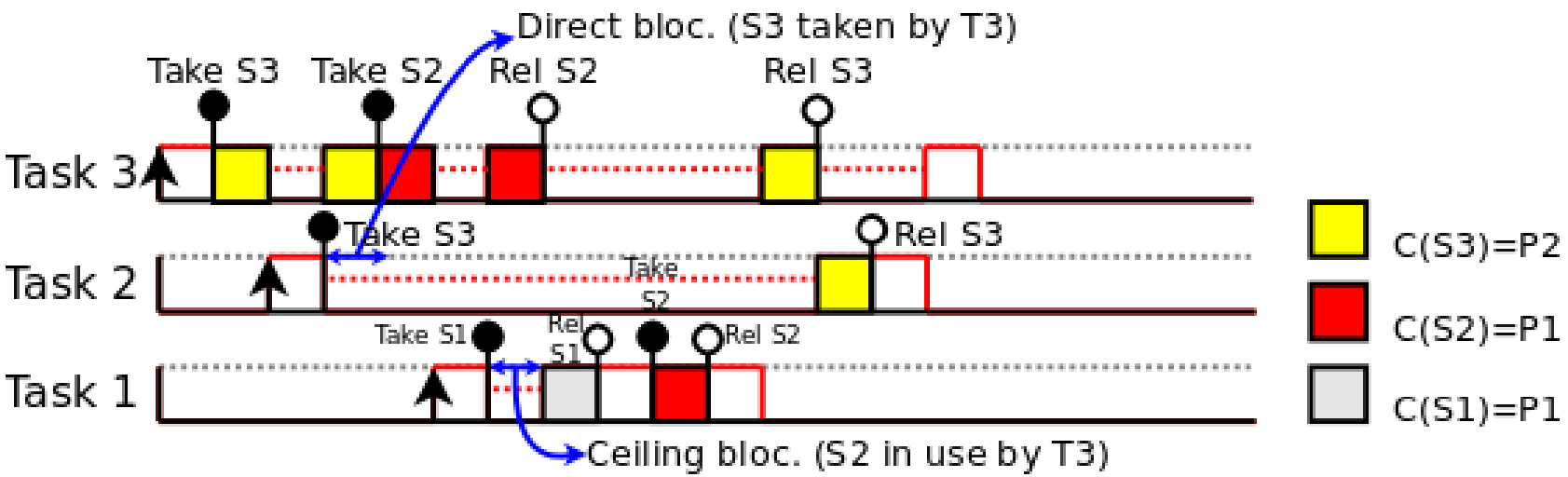
+ Relatively easy to implement
  - Only one additional field on the TCB, the inherited priority
+ Transparent to the programmer
  - Each task only uses local information
- Suffers from chain blocking and does not prevent deadlocks





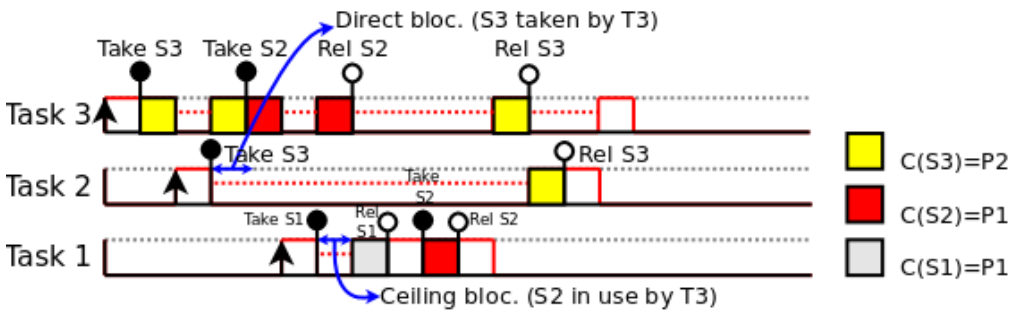
PCP – Priority Ceiling Protocol

- Extension of PIP with one additional rule about access to free semaphores, inserted to guarantee that all required semaphores are free.
- For each semaphore it is defined a **priority ceiling** , which equals the priority of the maximum priority task that uses it.
- A task can only **take a semaphore** if this one is free and if its **priority** is **greater than the ceilings** of all semaphores currently taken.



PCP – Priority Ceiling Protocol

- The PCP protocol only allows the access to the first semaphore when **all** other semaphores that a task needs are **free**
- To bound the blocking time (B) note that a task can be **blocked only once by lower priority tasks** . Only lower priority tasks that use semaphores which have a ceiling at least equal to the higher priority task can cause blocking.
- Note also that each task can only be blocked once



τ	S_1	S_2	S_3	B_i
τ_1	1	2	0	?
τ_2	0	9	3	?
τ_3	8	7	0	?
τ_4	6	5	4	?

PCP – Priority Ceiling Protocol

Worst-case blocking times

τ	S_1	S_2	S_3
τ_1	1	2	0
τ_2	0	9	3
τ_3	8	7	0
τ_4	6	5	4

$\Rightarrow$

τ	C_i	T_i	B_i
τ_1	5	30	9
τ_2	15	60	8
τ_3	20	80	6
τ_4	20	100	0

Schedulability analysis with PCP

- Same equations as for PIP, **only B_i computes differently**

Utilization test, for Rate Monotonic

$$\forall_{1 \le i \le N} \sum_{h:P_h > P_i} \left(\frac{C_h}{T_h}\right) + \frac{C_i + B_i}{T_i} \le i \cdot (2^{\frac{1}{i}} - 1)$$

Response Time Analysis (Fixed Priority)

$$R_i(0) = C_i + B_i$$
$$R_i(m) = C_i + B_i + \sum_{h:P_h > Pi} \left\lceil \frac{R_i(m-1)}{T_h} \right\rceil C_h$$

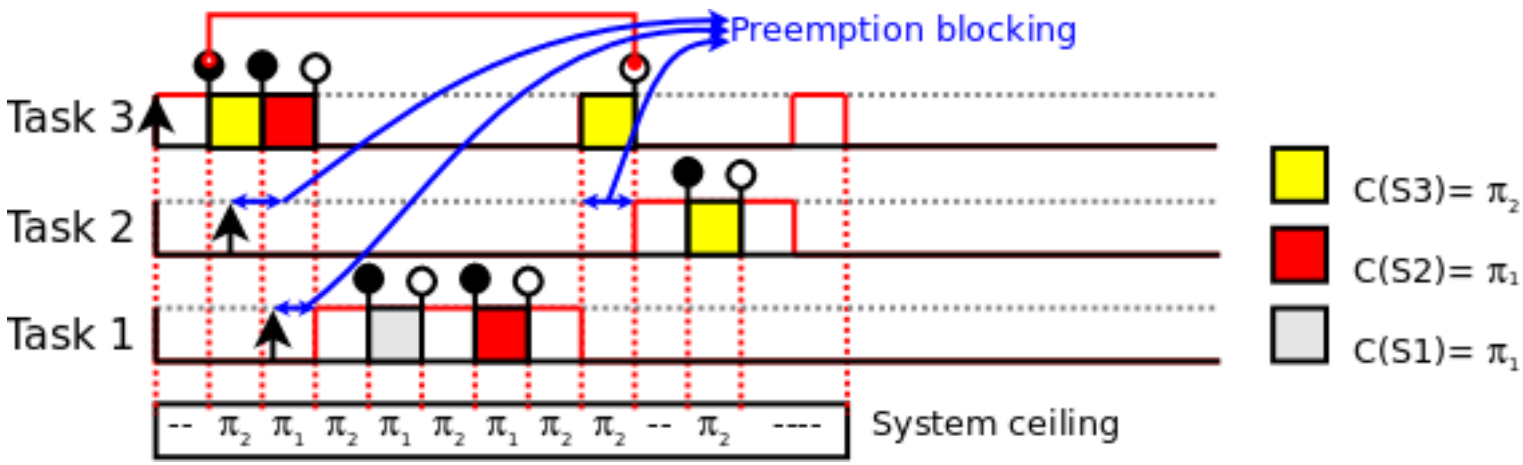
PCP – Priority Ceiling Protocol

PCP Evaluation

+ **Smaller blocking** than PIP, **free of chain blocking and deadlocks**
- Much **harder to implement** than PiP. On the TCB it requires one additional field for the inherited priority and another one for the semaphore where the task is blocked. To facilitate the transitivity of the inheritance it also requires a structure to the semaphores, their respective ceilings and the identification of the tasks that are using them
- Moreover, it is **not transparent to the programmer** as the semaphore ceilings are not local to the tasks

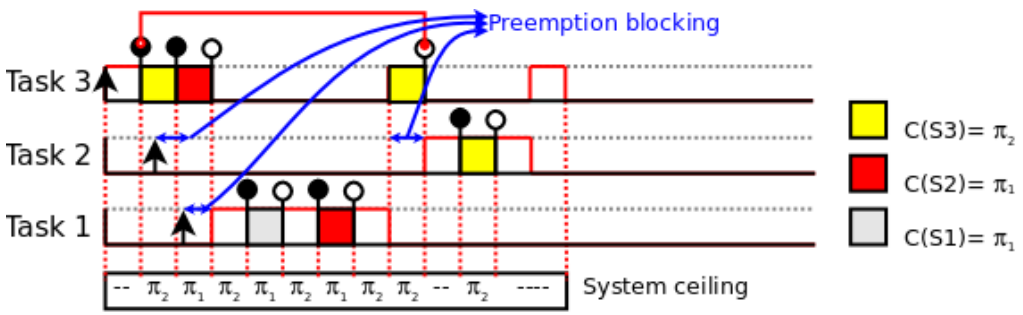
SRP – Stack Resource Policy

- Similar to PCP, but with one rule about the **beginning of execution** , to guarantee that all required semaphores are free
- Uses also the concept of priority ceiling
- Defines the preemption level (π) as the capacity of a task to cause preemption on another one (static parameter).
- A task may only start executing when its own **preemption level is higher than** the one of the executing task and also higher than the ceilings of all the semaphores in use (**system ceiling**).



SRP – Stack Resource Policy

- The SRP protocol only allows that a task **starts executing** when all **resources that it needs are free**
- The upper bound of the blocking time (B) is equal to the one of the PCP protocol, but it occurs in a different time - at the beginning of the execution instead of at the shared resource access.
- Each task can block only once by any task with a lower preemption level that uses a semaphore whose ceiling is at least equal to its preemption level.



τ	S_1	S_2	S_3	B_i
τ_1	1	2	0	?
τ_2	0	9	3	?
τ_3	8	7	0	?
τ_4	6	5	4	?

SRP – Stack Resource Policy

Worst-case blocking time

τ	S_1	S_2	S_3
τ_1	1	2	0
τ_2	0	9	3
τ_3	8	7	0
τ_4	6	5	4

$\Rightarrow$

τ	C_i	T_i	B_i
τ_1	5	30	9
τ_2	15	60	8
τ_3	20	80	6
τ_4	20	100	0

Schedulability analysis with SRP

- Same equations as for PIP and PCP. B_i **computes as for PCP**

Utilization test, for Rate Monotonic

$$\forall_{1 \le i \le N} \sum_{h:P_h > P_i} \left(\frac{C_h}{T_h}\right) + \frac{C_i + B_i}{T_i} \le i \cdot \left(2^{\frac{1}{i}} - 1\right)$$

Response Time Analysis (Fixed Priority)

$$R_i(0) = C_i + B_i$$
$$R_i(m) = C_i + B_i + \sum_{h:P_h > Pi} \left\lceil \frac{R_i(m-1)}{T_h} \right\rceil C_h$$

SRP – Stack Resource Policy

SRP Evaluation

+ Smaller blocking than PiP, free of chain blocking and deadlocks
+ Smaller number of preemptions than PCP, intrinsic compatibility with fixed an dynamic priorities and to resources with multiple units (i.e., that allow more than one concurrent access, e.g. buffer arrays)
- The hardest to implement (preemption test much more complex, requires computing the system ceiling, etc.)
- Not transparent to the programmer (semaphore ceilings, etc.)

Notes (1/2)

These policies are more complex to understand and implement than it seems! E.g.:

> "I observed in the kernel code (to my disgust), the Linux PIP implementation is a nightmare: extremely heavy weight, involving maintenance of a full wait-for graph, and requiring updates for a range of events, including priority changes and interruptions of wait operations."

> "An additional reason is that the original specification of PIP [24], despite being informally "proved" correct, is actually flawed.

Quoted from "Zhang, X., Urban, C. & Wu, C. Priority Inheritance Protocol Proved Correct. J Autom Reasoning 64, 73-95 (2020)."

Notes (2/2)

Equations in Buttazzo's book

- In the equations for computing the blocking times, Buttazzo's book has a "-1" factor affecting the critical region duration
- This is due to the fact that he considers a clock resolution of 1 t.u.
- This means that if a $t = t_k$ a resource is busy then it was taken at least at $t = t_k - 1$, so the task holding it already executed at least one time unit
- in these slides I'm considering that the clock resolution is much smaller than the duration of tasks and critical sections, so that factor is neglected.
- Other than that the equations are similar

Summary

- Access to shared resources: blocking
- The priority inversion problem: need to bound and analyze
- Basic techniques to allow exclusive access to shared resources
  - Disable interrupts, disable preemption
- Advanced techniques to allow exclusive access to shared resources
  - The Priority Inheritance Protocol – PIP (Linux, Xenomai, VXWorks...)
  - The Priority Ceiling Protocol – PCP (Mostly academic)
  - The Stack Resource Protocol - SRP (Mostly academic)