# Eingebettete Systeme Übungsblatt 10: PCP in RTEMS

Ingo Korb

LS 12, TU Dortmund

18.01.2016

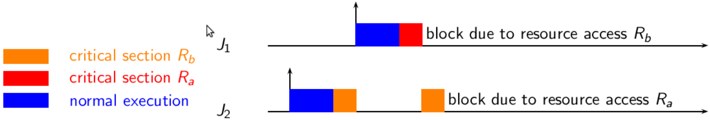
# Outline

- Drawback of PIP
- Introduction of PCP
- PCP theory / implementation
- Exercises

# Drawback?

Now we have PIP to mitigate the priority inversion and prevent the starvation of higher priority tasks. However, there is still a drawback:

- PIP might cause a *deadlock* if there are multiple resources




However, if the resource accesses for a task are properly nested, then some analysis is still possible.

- all the required semaphores are locked at once, or
- only one semaphore is used to guard one critical section, or
- a critical section guarded by a semaphore is completely within another critical section guarded by another semaphore with a predefined access order, or
- other ways to prevent deadlocks.

# Other ways? Priority Ceiling Protocol (PCP)

- Two key assumptions:
  - The assigned priorities of all jobs are fixed.
  - The resources required by all jobs are known a priori before the execution of any job begins.
- Definition: The *priority ceiling* of a resource $R$ is the highest priority of all the jobs that require $R$, and is denoted $\Pi(R)$.
- Definition: The *current priority* of a job $J$ at time $t$ is denoted $\pi(t, J)$, initialized to the jobs priority level when $J$ is released. (smaller means higher priority)
- Definition: The *current priority ceiling* $\Pi'(t)$ of the system is equal to the highest priority ceiling of the resources currently in use at time $t$, or $\Omega$ if no resources are currently in use. $\Omega$ is a priority lower than any real priority.
- Use the priority ceiling to decide whether a higher priority can allocate a resource or not.

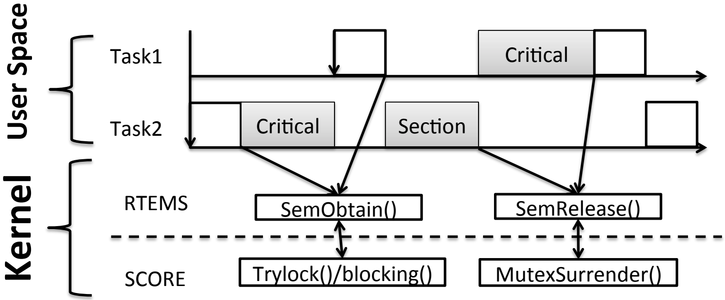
# Theoretical PCP

1. Scheduling Rule
   - Every job $J$ is scheduled based on the current priority $\pi(t, J)$.
2. Allocation Rule: Whenever a job $J$ requests a resource R at time $t$, one of the following two conditions occurs:
   - $R$ is held by another job and $J$ becomes blocked.
   - $R$ is free:
     - If $J$'s priority $\pi(t, J)$ is higher than the current priority ceiling $\Pi'(t)$, $R$ is allocated to $J$.
     - Otherwise, only if $J$ is the job holding the resource(s) whose priority ceiling equals $\Pi'(t)$, $R$ is allocated to $J$
     - Otherwise, $J$ becomes blocked.
3. Priority-inheritance Rule: When $J$ becomes blocked, the job $J_l$ that blocks $J$ inherits the current priority $\pi(t, J)$ of J. $J_l$ executes at its inherited priority until it releases every resource whose priority ceiling is $\geq \pi(t, J)$ (or until it inherits an even higher priority); at that time, the priority of $J_l$ returns to its priority $\pi(t', J_l)$ at the time $t'$ when it was granted the resources.

# PCP Implementation

1. Scheduling Rule
   - Every job $J$ is scheduled based on the current priority $\pi(t, J)$.
2. Allocation Rule: Whenever a job $J$ requests a resource R at time $t$, one of the following two conditions occurs:
   - $R$ is held by another job and $J$ becomes blocked.
   - $R$ is free:
     - If $J$'s priority $\pi(t, J)$ equals the semaphore ceiling $\Pi(R)$, $R$ is allocated to $J$ directly.
     - If priority $\pi(t, J)$ is lower than $\Pi(R)$, $R$ is allocated to $J$. Moreover, priority $\pi(t, J)$ needs to be raised to ceiling $\Pi(R)$.
     - If priority $\pi(t, J)$ is larger than $\Pi(R)$. (Won't normally happen.)
3. When releasing the semaphore, $J$ returns to its priority $\pi(t', J)$, then further check if another job $J_l$ was waiting for this semaphore. If so, transfer the semaphore to locked $J_l$ and raise it's priority to the semaphore priority ceiling $\Pi(R)$.

# Overview of PIP/PCP




1. In SemObtain() of RTEMS, it will call _CORE_mutex_Seize().
2. In SemRelease() of RTEMS, it will call _CORE_mutex_Surrender().
3. After calling the above interface functions in SCORE, the mutex-relevant features will be triggered.

# Exercises (10 points)

1. Please activate PIP for the DOUBLE_SEMAPHORE example and see whether it does work. Draw the scheduling diagram. (4 points)
2. Please complete the missing code of the PCP implementation to get rid of the deadlock due to the resource competition. The relevant files are: coremuteximpl.h, coremutexsurrender.c (6 points)
   Hints: Please refer slides 6 and 7 to fulfil the code.
3. _Thread_Raise_priority(A, B), raises the priority of A to B.
4. _Thread_queue_Release( queue, lock ), releases the lock from the wait queue of a mutex.

# Task data

| Tasks | Period | Critical Section | Arrive Time | Semaphore |
|---|---|---|---|---|
| $\tau_1$ | 20 | 6 | 2 | II |
| $\tau_2$ | 30 | 0 | 5 | X |
| $\tau_3$ | 40 | 6 | 0 | I |