Spring 2009 - Real-Time Systems

http://www.neu-rtes.org/courses/spring2009/

Chapter 6
Resource Sharing
In Real-Time Systems

Real-Time Embedded Systems Laboratory
Northeastern University

Objectives

- In this chapter, you are supposed to learn:
  - What are the major problems of resource sharing in real-time systems
  - What are the basic ideas to resolve the problems
  - How does PIP work?
  - How does PCP work?

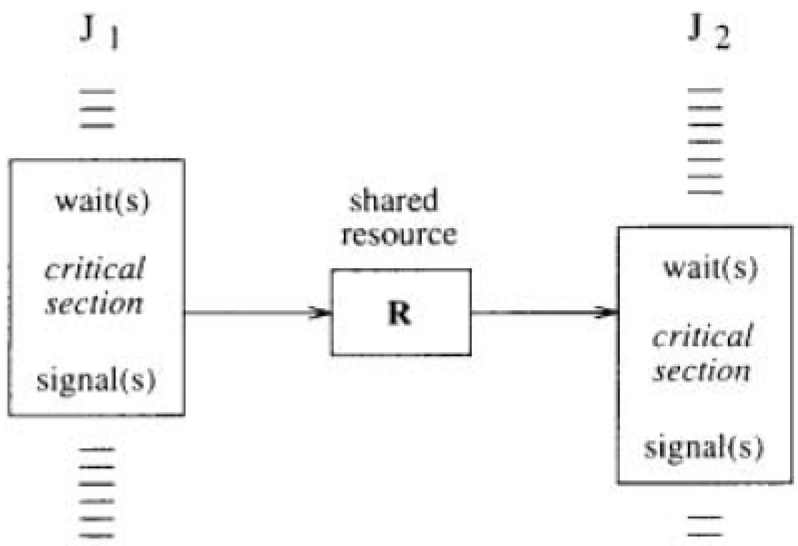
Contents

- Resource Sharing Problems
- Resource Access Policies
  - Non-Preemptive Protocol (NPP)
  - Highest Locker Priority (HLP)
  - Priority Inheritance Protocol (PIP)
  - Priority Ceiling Protocol (PCP)
- Schedulability Test under PCP

Resource Sharing Model

Examples of common resources: data structures, variables, main memory area, file, set of registers, I/O unit, … .

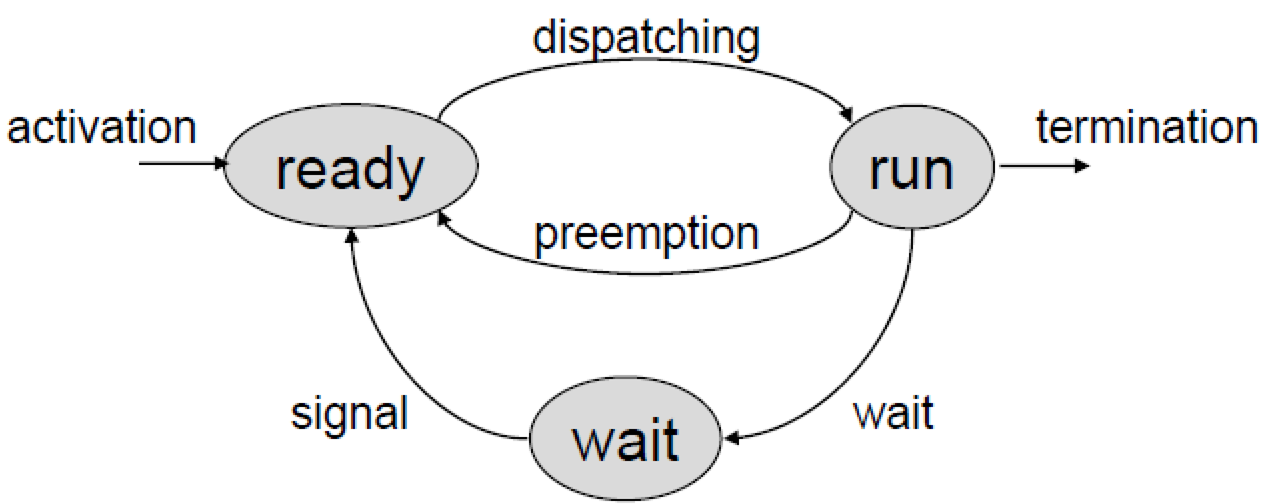
Many shared resources do not allow simultaneous accesses but require ***mutual exclusion*** (***exclusive resources***). A piece of code executed under mutual exclusion constraints is called a ***critical section***.



Resource Sharing Model

A task waiting for an exclusive resource is said to be ***blocked*** on that resource. Otherwise, it proceeds by entering the ***critical section*** and ***holds*** the resource. When a task leaves a critical section, the associated resource becomes ***free***.

Waiting state caused by resource constraints:



Resource Sharing Model

Each ***exclusive resource*** R_i must be protected by a different ***semaphore*** S_i and each critical section operating on a resource must begin with a *wait*(S_i) primitive and end with a *signal*(S_i) primitive.

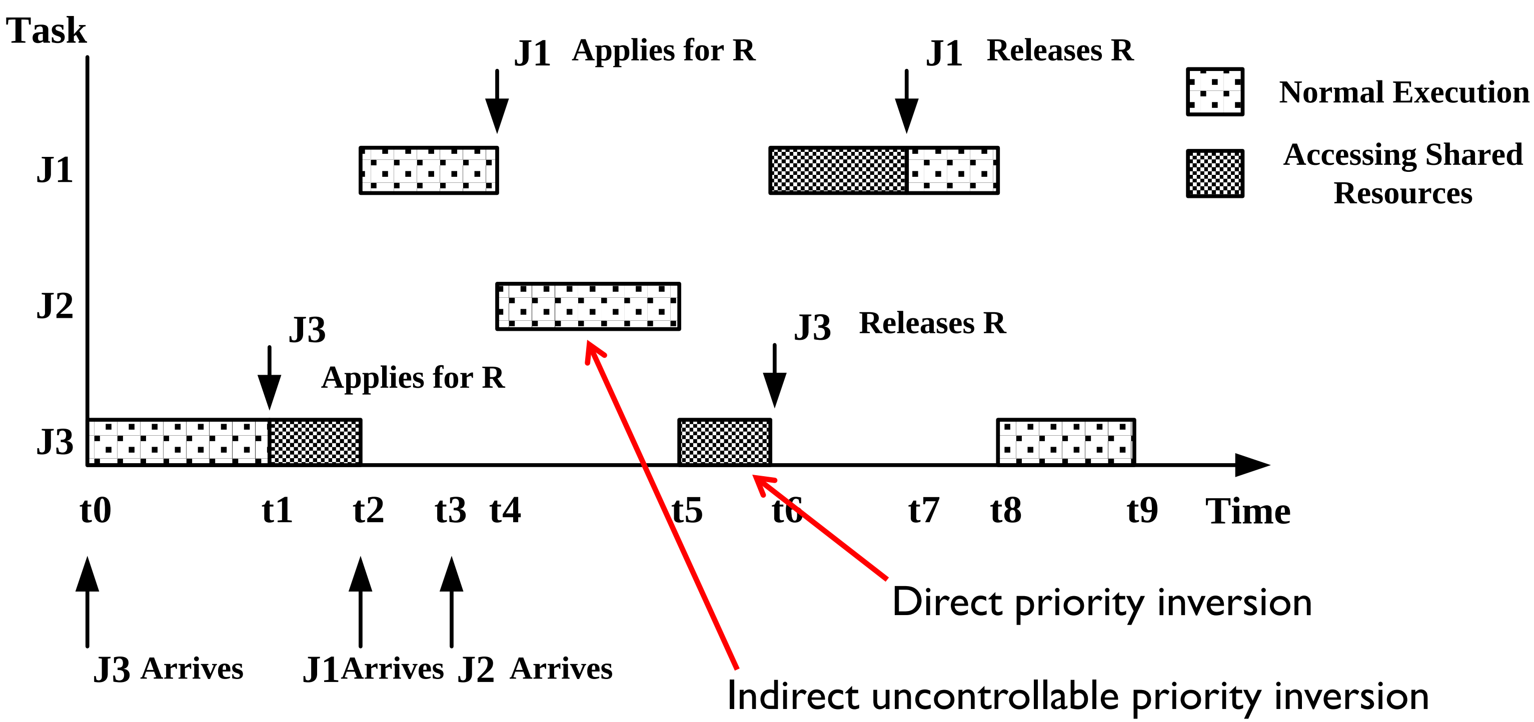
All tasks blocked on the same resource are kept in a queue associated with the semaphore. When a running task executes a ***wait*** on a ***locked semaphore***, it enters a ***waiting state***, until another tasks executes a ***signal*** primitive that ***unlocks the semaphore***.

Resource Sharing in GPOS

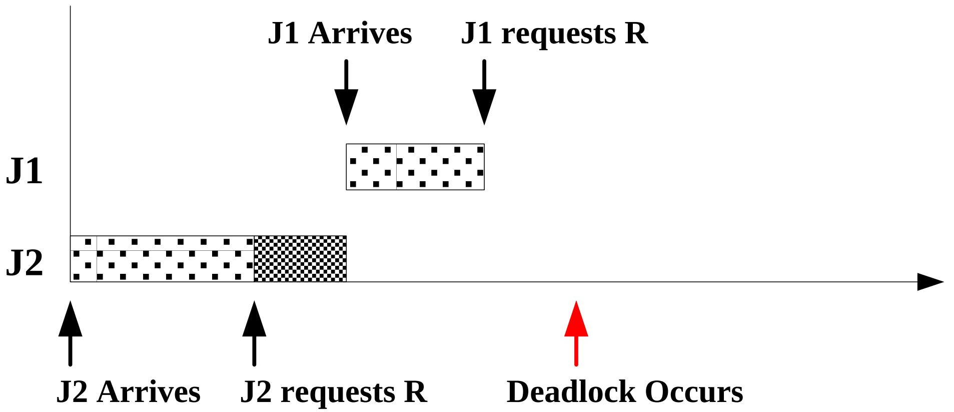
- Resource Sharing Issues in GPOS
  - Data Consistency
    - Semaphores and Monitors are used to guarantee data consistency
  - Deadlock
    - Deadlock prevention methods (Resource ordering)
    - Deadlock breaking methods
- Incapability of Policies in GPOS
  - Only logical results are taken into consideration
  - No bounded time on resource accessing
  - Ignorant of priorities of tasks
  - Un-predictable blocking behaviors

Priority Inversion

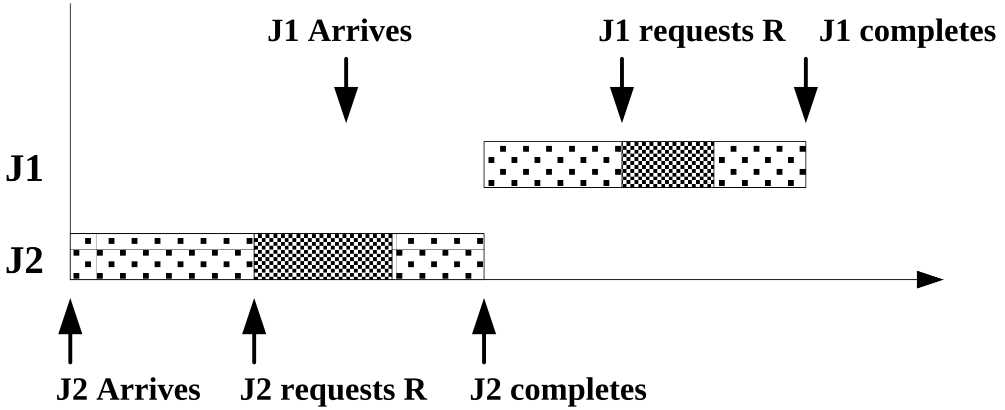
- Preemption + Priority-Based → Priority Inversion



Deadlock Still Exists



(a) In preemption mode
Deadlock occurs



(b) In non-preemption mode
No deadlock occurs

Contents

- Resource Sharing Problems
- Resource Access Policies
  - Non-Preemptive Protocol (NPP)
  - Highest Locker Priority (HLP)
  - Priority Inheritance Protocol (PIP)
  - Priority Ceiling Protocol (PCP)
- Schedulability Test under PCP

Resource Access Control

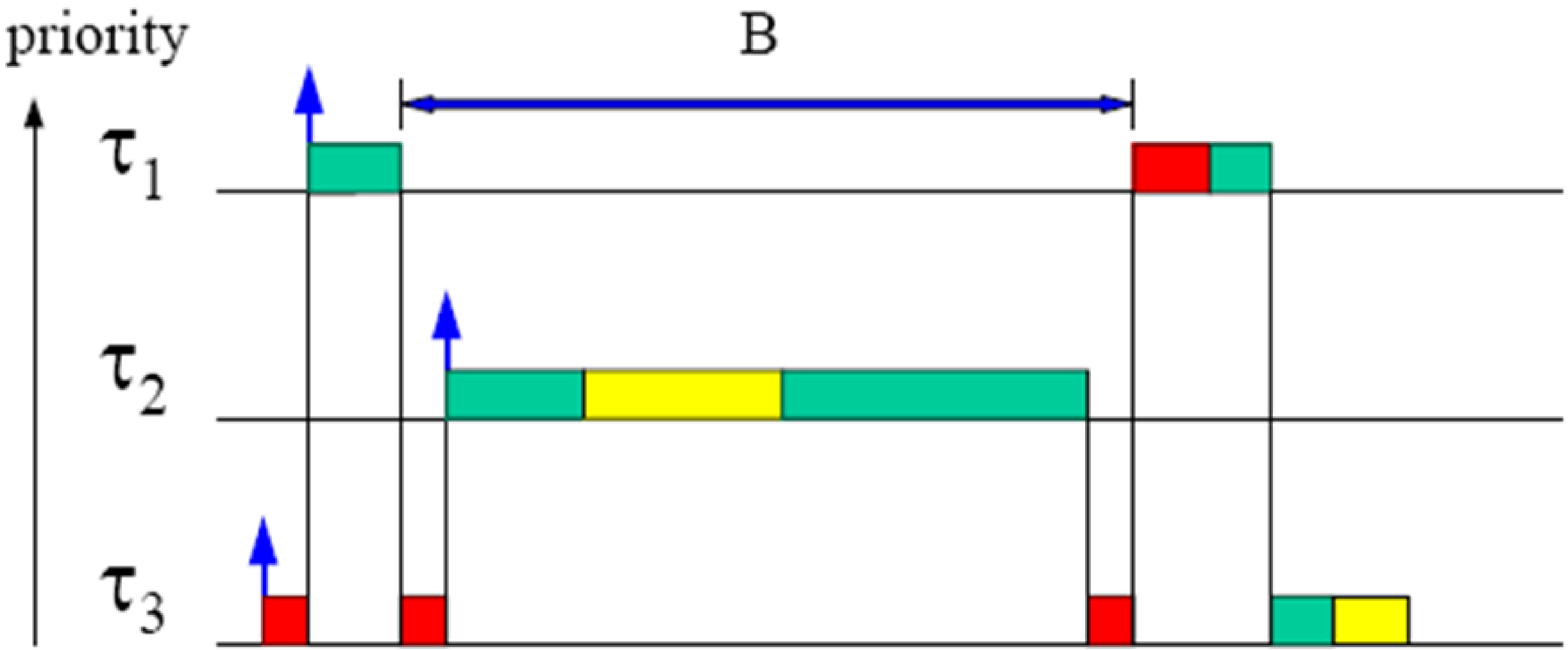
- Under Fixed Priorities
  - Non-Preemptive Protocol (NPP)
  - Highest Locker Priority (HLP)
  - Priority Inheritance Protocol (PIP)
  - Priority Ceiling Protocol (PCP)
- Under Dynamic Priorities
  - Stack Resource Policy (SRP)

Non-Preemptive Protocol

- Basic Idea: Preemption is forbidden in critical sections
- Implementation: when a task enters a CS, its priority is raised to the highest value
- Advantage: simplicity
- Problems: High priority tasks that do not use CS may also block

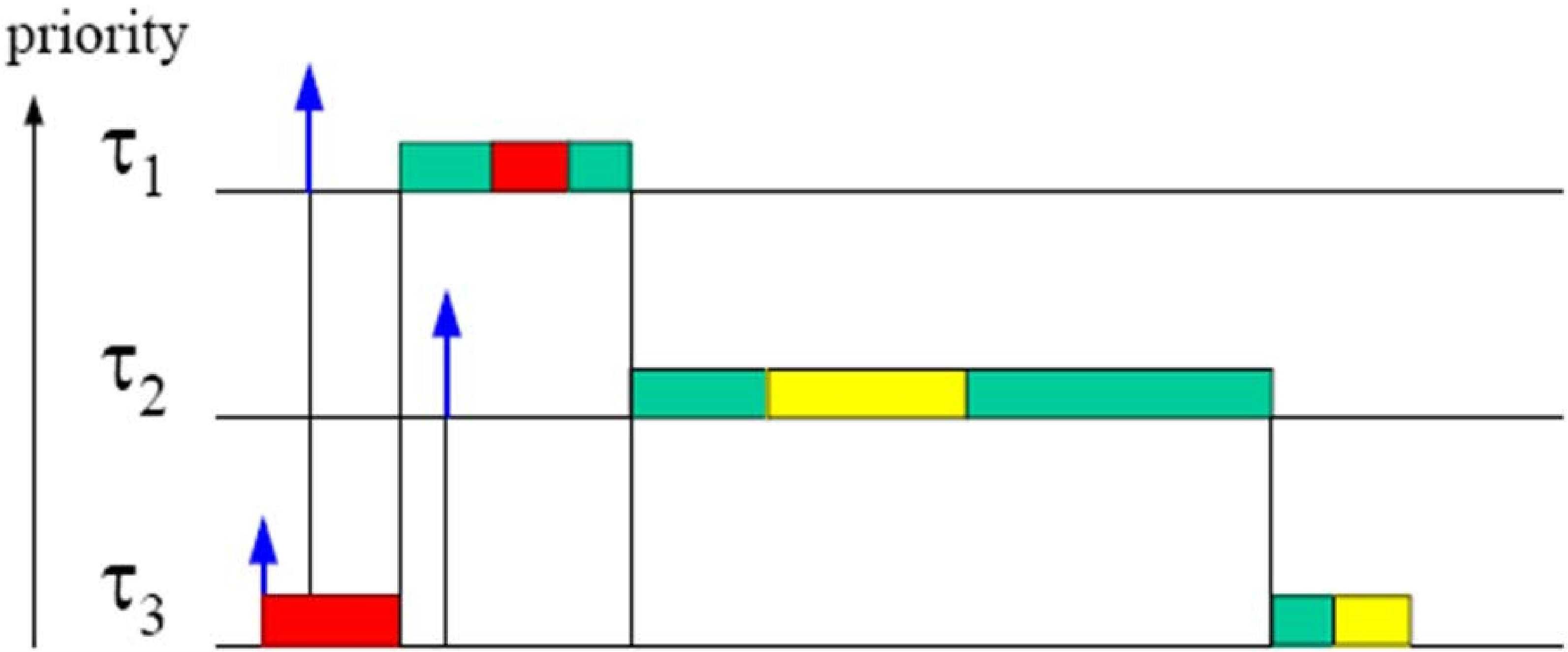
Non-Preemptive Protocol

- With Preemption in CS



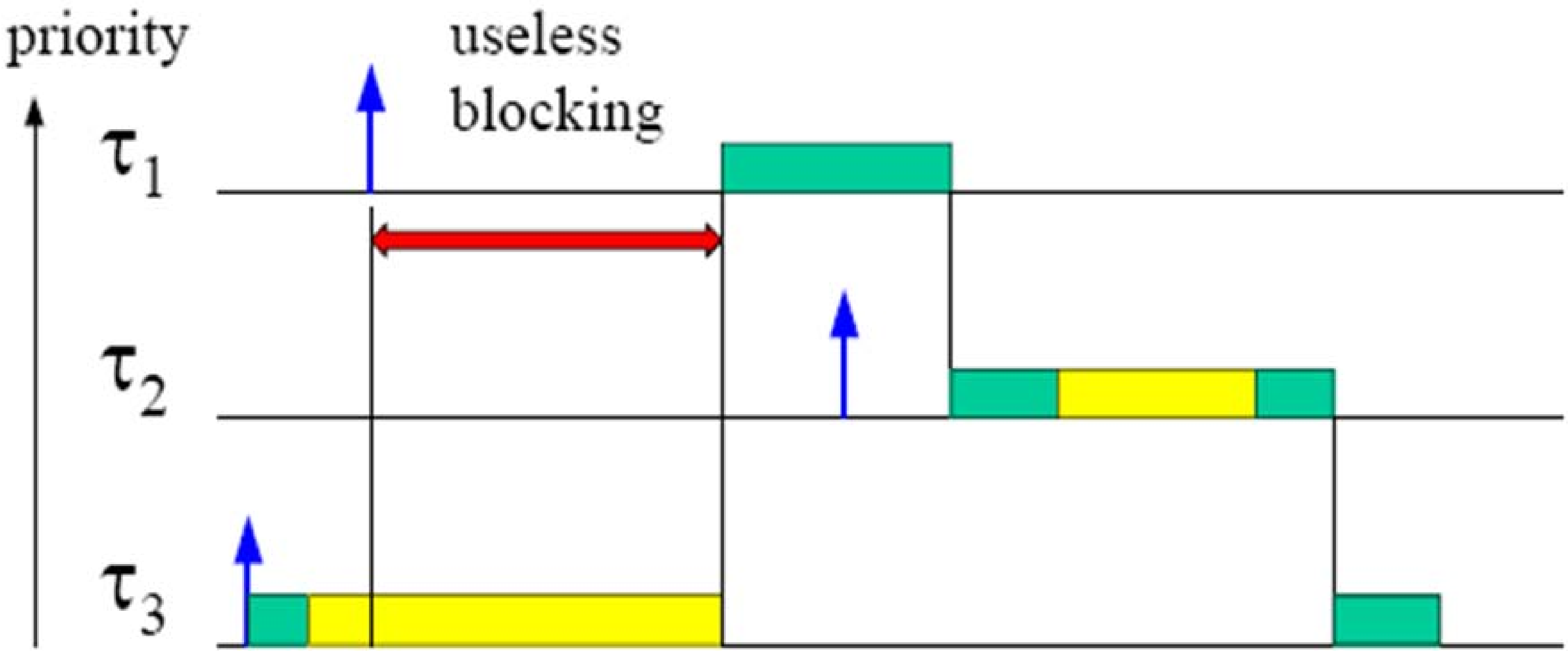
Non-Preemptive Protocol

- Without Preemption in CS



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

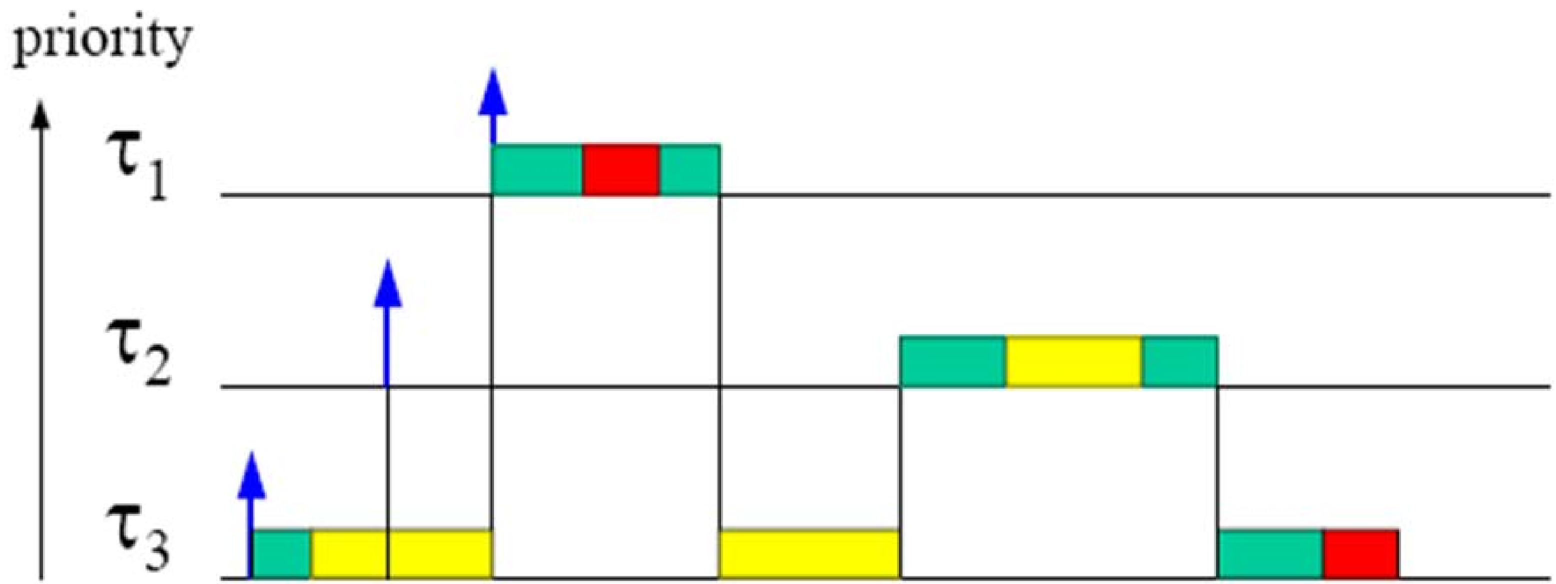
Problems with NPP



τ_1 cannot preemt, although it could

Highest Locker Priority

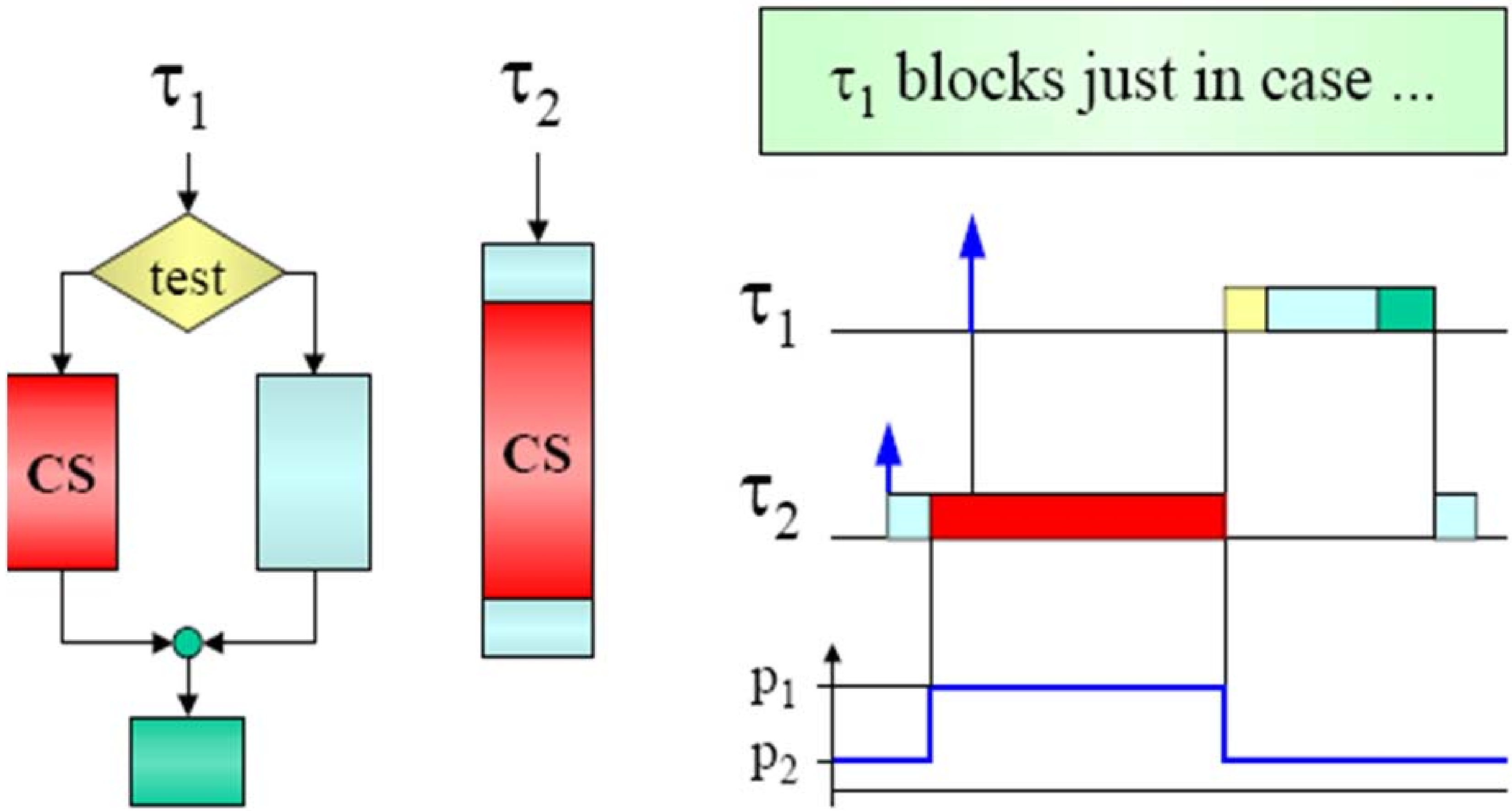
Schedule with HLP



$$P_{CS} = \max \{P_k \mid \tau_k \text{ uses CS}\}$$

τ_2 is blocked, but τ_1 can preempt within a CS

Problems with HLP



Contents

- Resource Sharing Problems
- Resource Access Policies
  - Non-Preemptive Protocol (NPP)
  - Highest Locker Priority (HLP)
  - Priority Inheritance Protocol (PIP)
  - Priority Ceiling Protocol (PCP)
- Schedulability Test under PCP

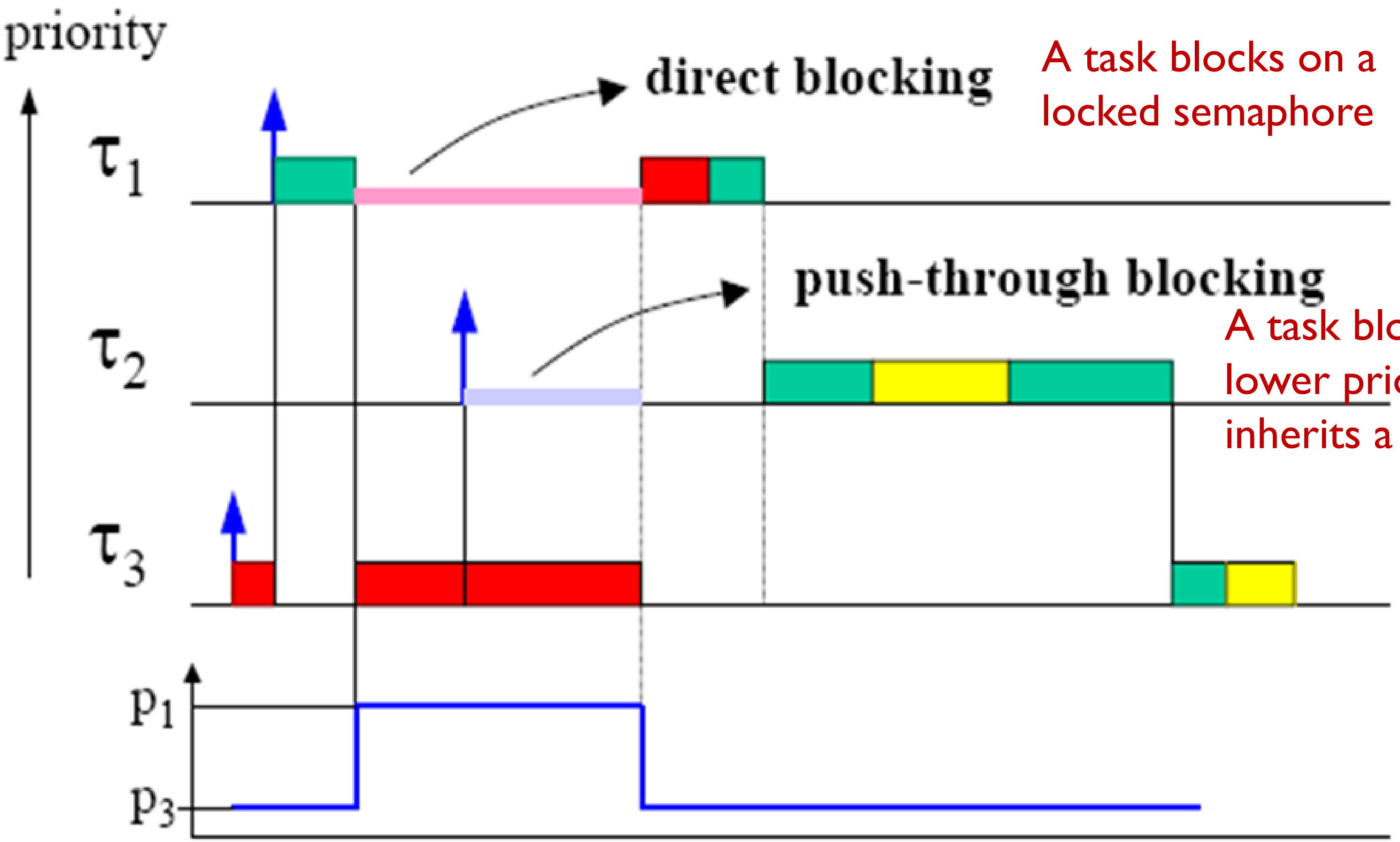
Priority Inheritance Protocol (PIP)

- Basic Idea: When a task Ji blocks one or more higher priority tasks, it temporarily assumes (inherits) the highest priority of the blocked tasks. When *J* exits the critical section, it must resume the priority it had when entering the CS

- Priority inheritance is transitive. For instance, suppose J_1, J_2 and J_3 are assigned priority in descending order, if J_3 blocks J_2, and J_2 blocks J_1, then J_3 will inherit the priority of J_1

- A job J can preempt another job J_L is job J is not blocked and its priority is higher than the priority, inherited or assigned, at which job J_L is executing

Schedule with PIP



A task blocks on a locked semaphore

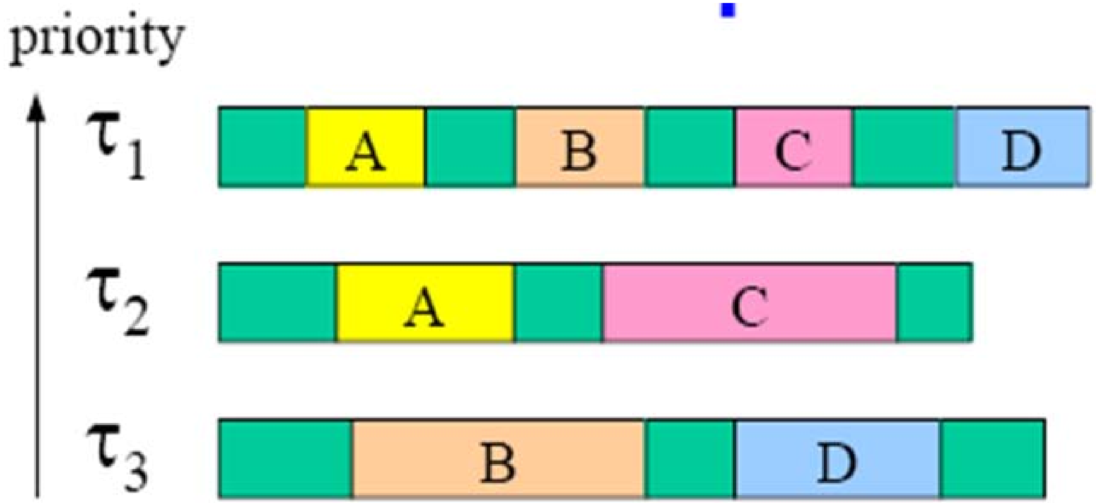
A task blocks because a lower priority task inherits a higher priority

Properties of PIP

- Property 1: A task can be blocked at most once by each lower priority task
- Property 2: A task can be blocked at most once by each semaphore it accesses

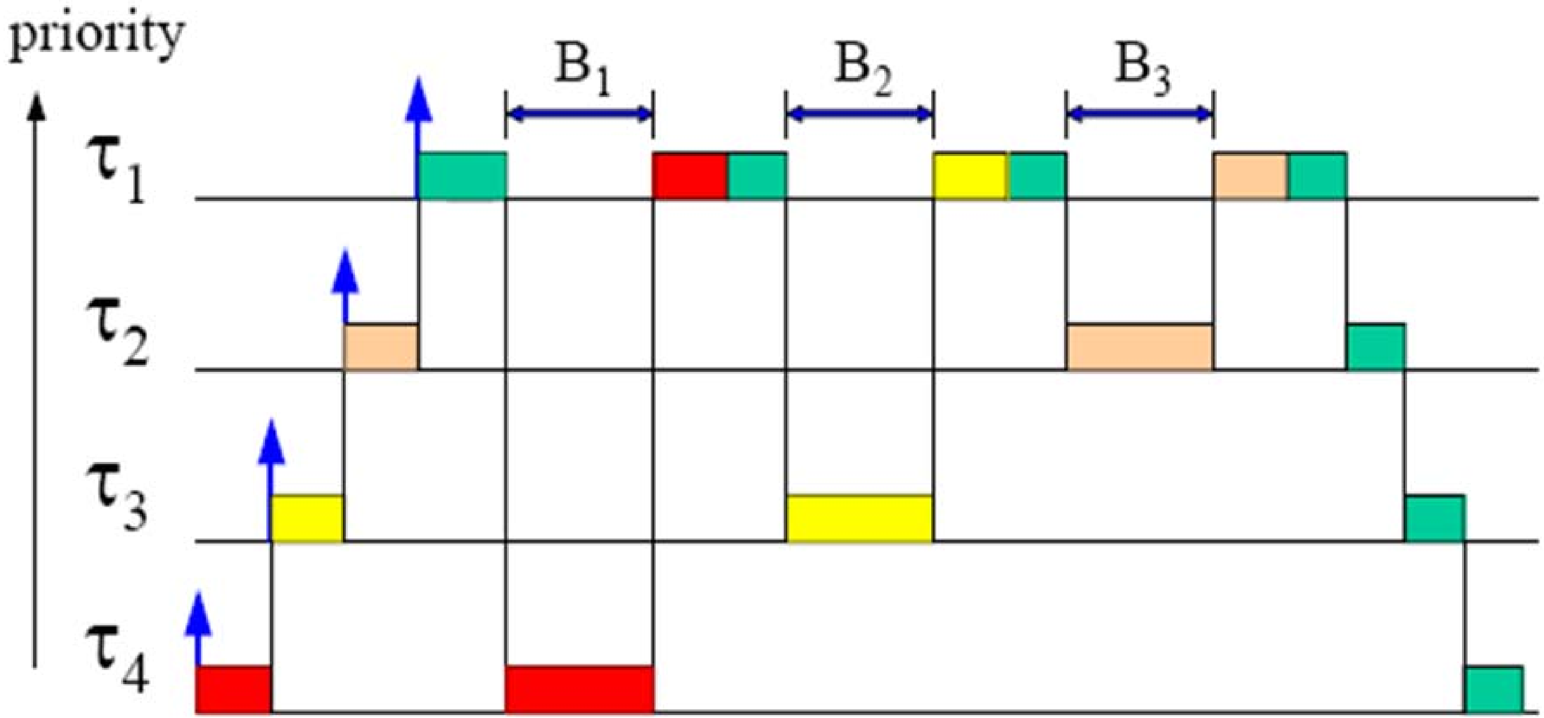
- If n is the number of lower priority tasks of t_i, and m is the number of semaphores that t_i can be blocked, then t_i can be blocked at most for the duration of min(n, m) critical sections

PIP Properties – An Example



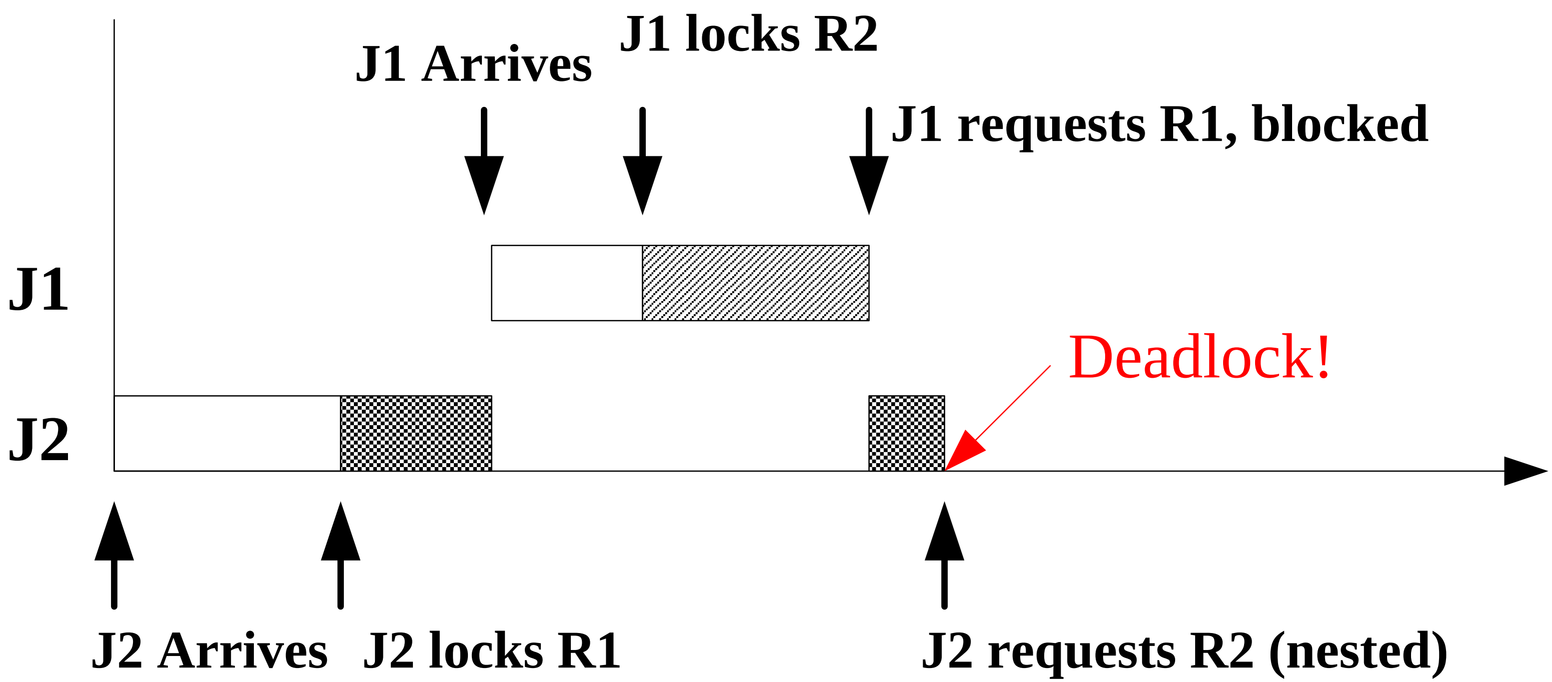
- τ_1 can be blocked once by τ_2 (on A_2 or C_2) and once by τ_3 (on B_3 or D_3)
- τ_2 can be blocked once by τ_3 (on B_3 or D_3)
- τ_3 cannot be blocked

Problem 1: Chained Blocking



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

Problem 2: Deadlock still Exists



Contents

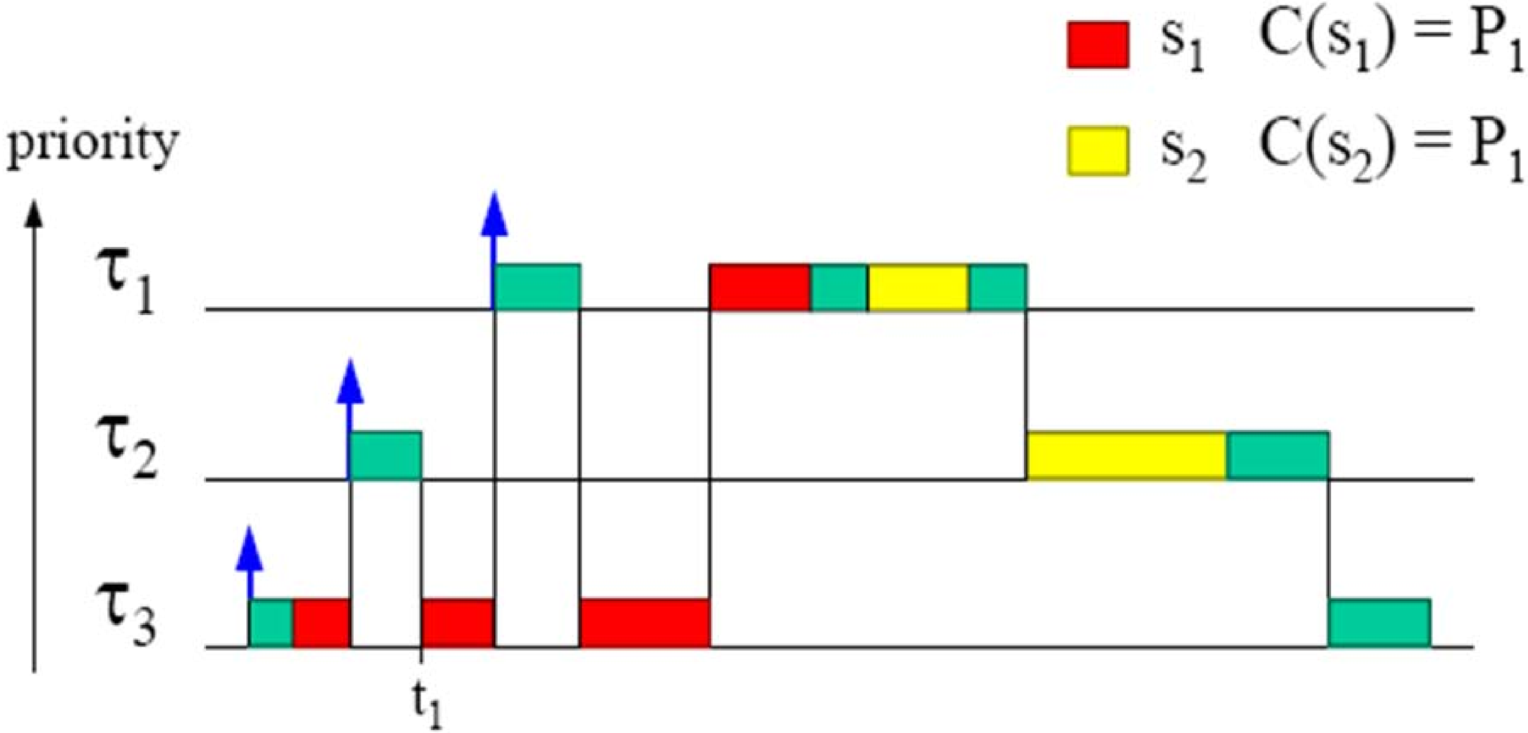
Priority Ceiling Protocol (PCP)

- The goal of PCP protocol is to avoid deadlock and chained blocking

- Basic Idea: To ensure that when a job J preempts the critical section of another job and executes its own critical section z, the priority at which z will executes is guaranteed to be higher than the inherited priorities of all the preempted critical sections.

- The idea is realized by firstly assigning a priority to each semaphore, which is equal to the highest priority task that may use this semaphore. A job J can start its execution in critical section only if J's priority is higher than all priority ceilings of all the semaphores locked by jobs other than J.

Schedule with PCP

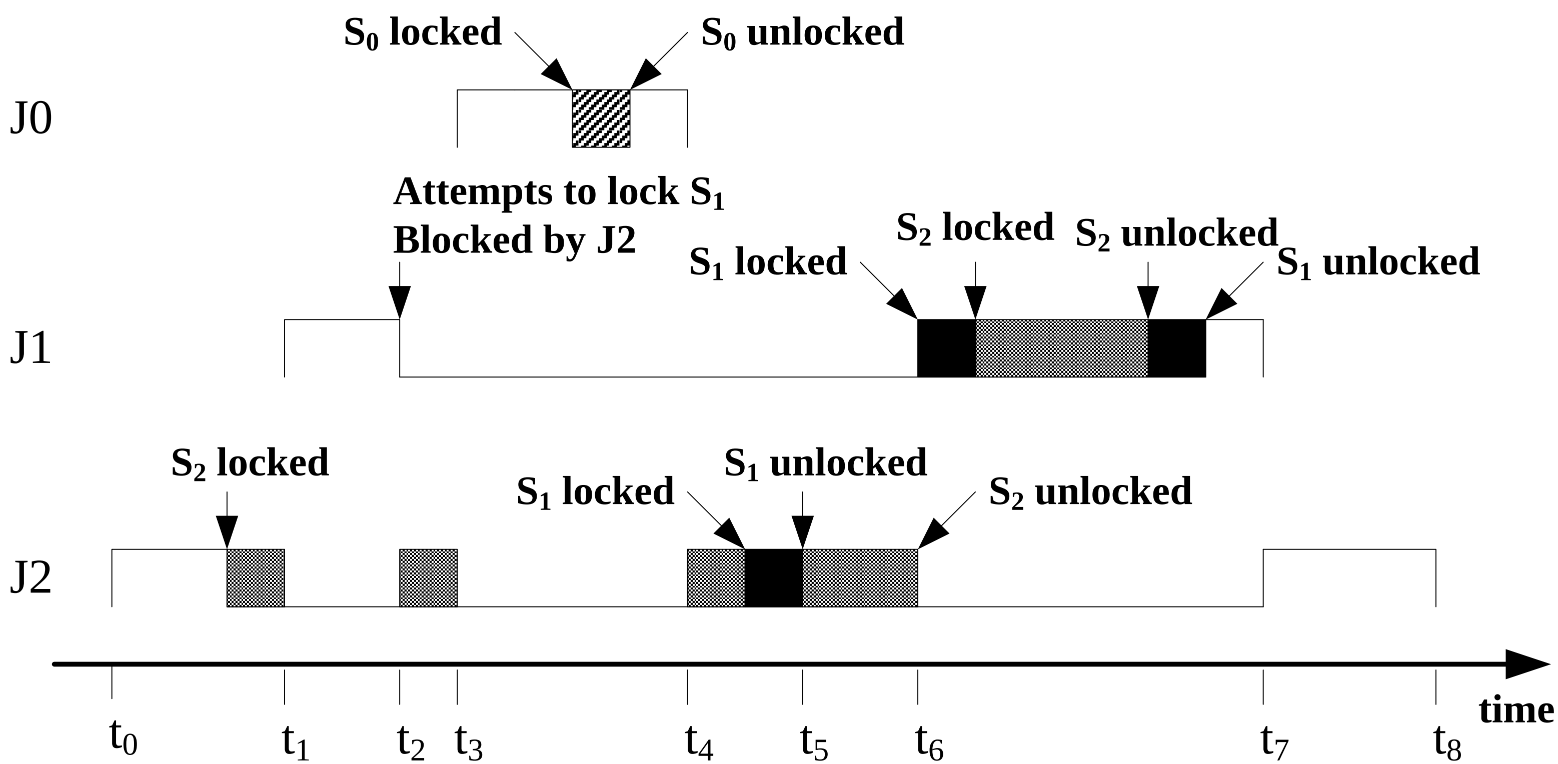


t_1: τ_2 is blocked by the PCP, since $P_2 < C(s_1)$

Avoiding Deadlock by PCP

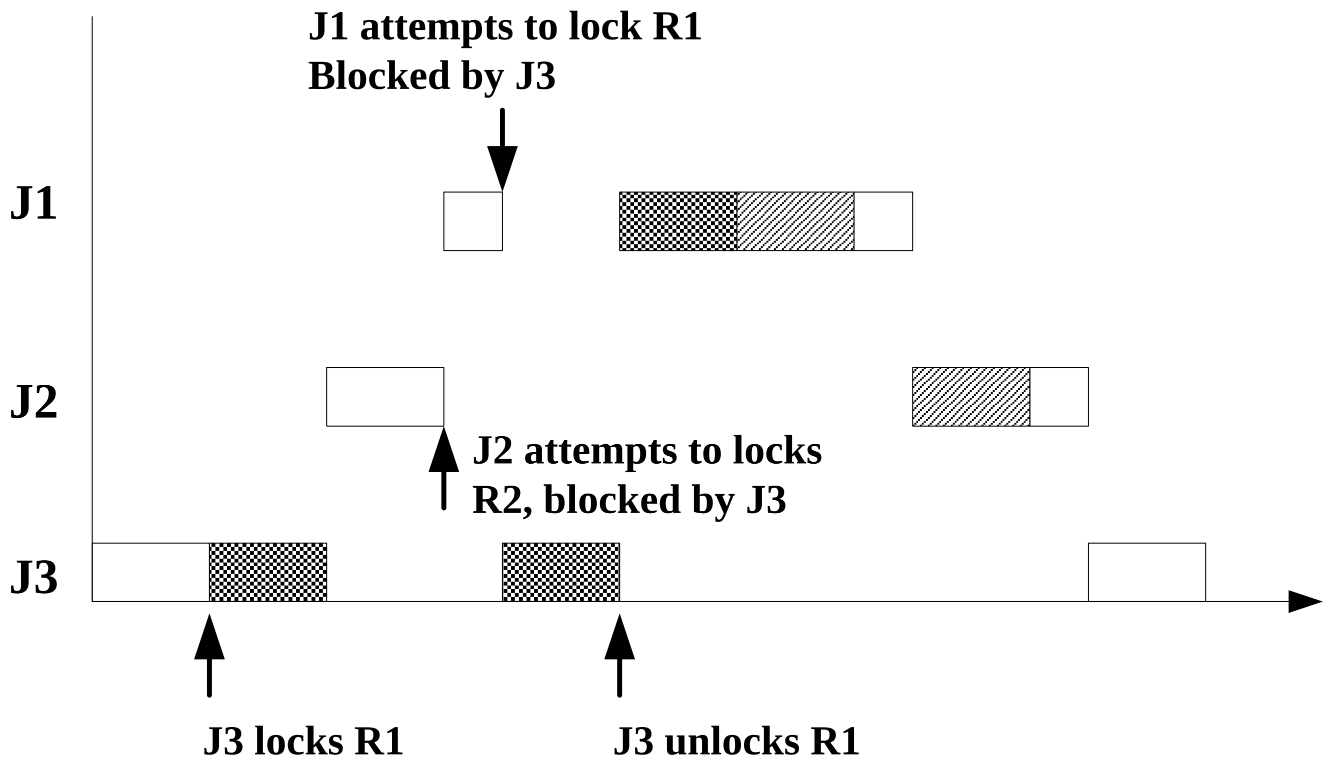
- An Example
  - Task properties
    - $J_0 = \{\ldots, P(S_0), \ldots, V(S_0), \ldots\}$
    - $J_1 = \{\ldots, P(S_1), \ldots, P(S_2), \ldots, V(S_2), \ldots, V(S_1), \ldots\}$
    - $J_2 = \{\ldots, P(S_2), \ldots, P(S_1), \ldots, V(S_1), \ldots, V(S_2), \ldots\}$
  - Priority ceilings of semaphores
    - $P_{S0} = \max\{P_0\} = P_0$
    - $P_{S1} = \max\{P_1, P_2\} = P_1$
    - $P_{S2} = \max\{P_1, P_2\} = P_1$

Avoiding Deadlock by PCP



Avoiding Chained Blocking by PCP

- Assume J_1 access S_1 and S_2, J_2 accesses S_2 and J_3 accesses S_1
- According to PCP, $P_{S1} = P_{S2} = P_{J1}$



Properties of PCP

- Property 1: PCP can avoid deadlock
- Property 2: Blocking is reduced to only one CS
- PCP protocol has the "*at-most-once*" property, which is highly desired in timing analysis

- Problem: PCP is not transparent to programmers - semaphores needs manual ceiling (review PIP, inheritance can be done without user intervention)

Contents

- Resource Sharing Problems
- Resource Access Policies
  - Non-Preemptive Protocol (NPP)
  - Highest Locker Priority (HLP)
  - Priority Inheritance Protocol (PIP)
  - Priority Ceiling Protocol (PCP)
- Schedulability Test under PCP

RM Schedulability Test Extended

- A set of n periodic tasks using PCP can be scheduled by RM algorithms if the following conditions are satisfied

$$\forall i,\ 1 \le i \le n, \qquad \frac{C_1}{T_1} + \frac{C_2}{T_2} + \cdots + \frac{C_i}{T_i} + \frac{B_i}{T_i} \le i(2^{1/i} - 1)$$

- A set of n periodic tasks using PCP protocol can be scheduled by RM algorithm if the following condition is satisfied

$$\frac{C_1}{T_1} + \cdots + \frac{C_n}{T_n} + \max\left(\frac{B_1}{T_1}, \cdots, \frac{B_{n-1}}{T_{n-1}}\right) \le n(2^{1/n} - 1)$$

- A set of n periodic tasks using PCP can be scheduled by RM algorithm for all task phasing if

$$\forall i,\ 1 \le i \le n,$$

$$\min_{(k,l)\in R_i} \left[\sum_{j=1}^{i-1} U_j \frac{T_j}{lT_k} \left\lceil \frac{lT_k}{T_j} \right\rceil + \frac{C_i}{lT_k} + \frac{B_i}{lT_k}\right] \le 1$$

Recommended Readings

1. Jane W.S. Liu, *Real-Time Systems*, 2002.
2. Liu Sha, R. Rajkumar and J.P. Lehoczky, *Priority Inheritance Protocols an approach to real-time synchronization*.
3. *Priority inversion why you care and what to do about it.*
4. N. Audsley and A. Burns, *Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling*.

- Acknowledgement
  - Lots of slides in this chapter are borrowed from Prof. Zonghua Gu's RTS course at HKUST, here we show our thankfulness to Prof. Gu ☺
  - http://www.cse.ust.hk/~zgu/comp680g/

Visit Our Website ☺

- The Website of Real-Time Embedded Systems Laboratory, Northeastern University
  - http://www.neu-rtes.org
  - http://www.neu-rtes.org/courses/spring2009/
- You can find
  - General information on the projects conducted in our lab
  - Research and publications
  - Research information and contacts of the members
  - Some useful research links
- Write me emails if you have questions in RTS
  - mingsong@research.neu.edu.cn