Sharif University of Technology
Department of Computer Science and Engineering

Lec. 7:

Resource Access Protocols

Real-Time Computing

S. Safari

2023

Introduction

- A resource is any software structure that can be used by a process to advance its execution.
  - A resource can be a data structure, a set of variables, a main memory area, a file, or a set of registers of a peripheral device.
  - A resource dedicated to a particular process is said to be *private*, whereas a resource that can be used by more tasks is called a *shared resource*.
- A shared resource protected against concurrent accesses is called an exclusive resource.
- To ensure consistency of the data structures in exclusive resources, any concurrent operating system should use appropriate resource access protocols to guarantee a mutual exclusion among competing tasks.

Introduction (2)

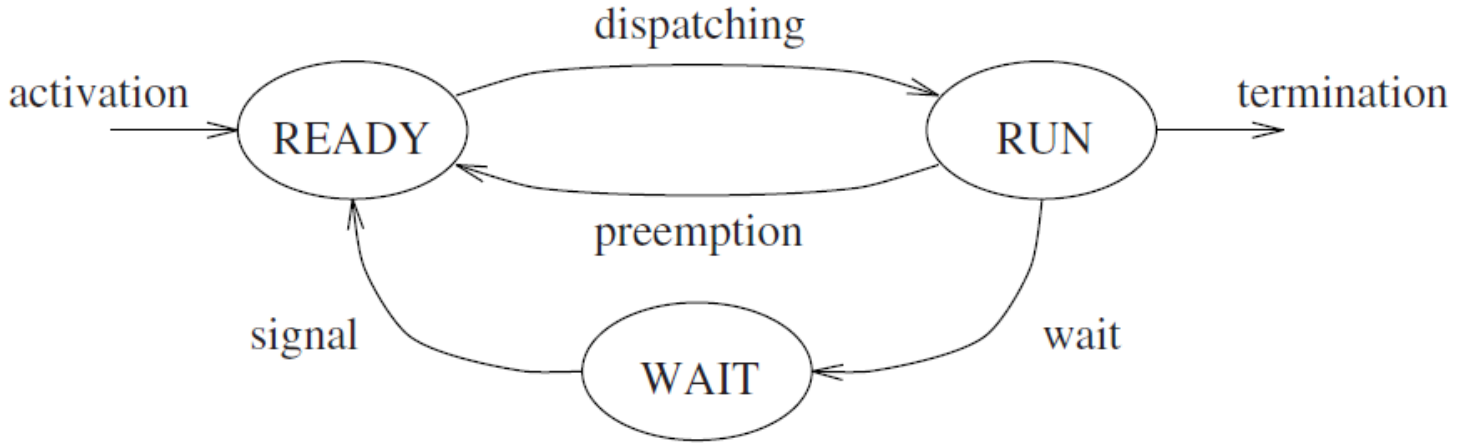
- A piece of code executed under mutual exclusion constraints is called a *critical section*.
- Any task that needs to enter a critical section must wait until no other task is holding the resource.
- A task waiting for an exclusive resource is said to be *blocked* on that resource, otherwise it proceeds by entering the critical section and holds the resource.
- When a task leaves a critical section, the resource associated with the critical section becomes free, and it can be allocated to another waiting task, if any.

Introduction (3)

- Operating systems typically provide a general synchronization tool, called a *semaphore*, that can be used by tasks to build critical sections.
- A semaphore is a kernel data structure that, apart from initialization, can be accessed only through two kernel primitives, usually called *wait* and *signal*.
- When using this tool, each exclusive resource R_k must be protected by a different semaphore S_k and each critical section operating on a resource R_k must begin with a wait(S_k) primitive and end with a signal(S_k) primitive.

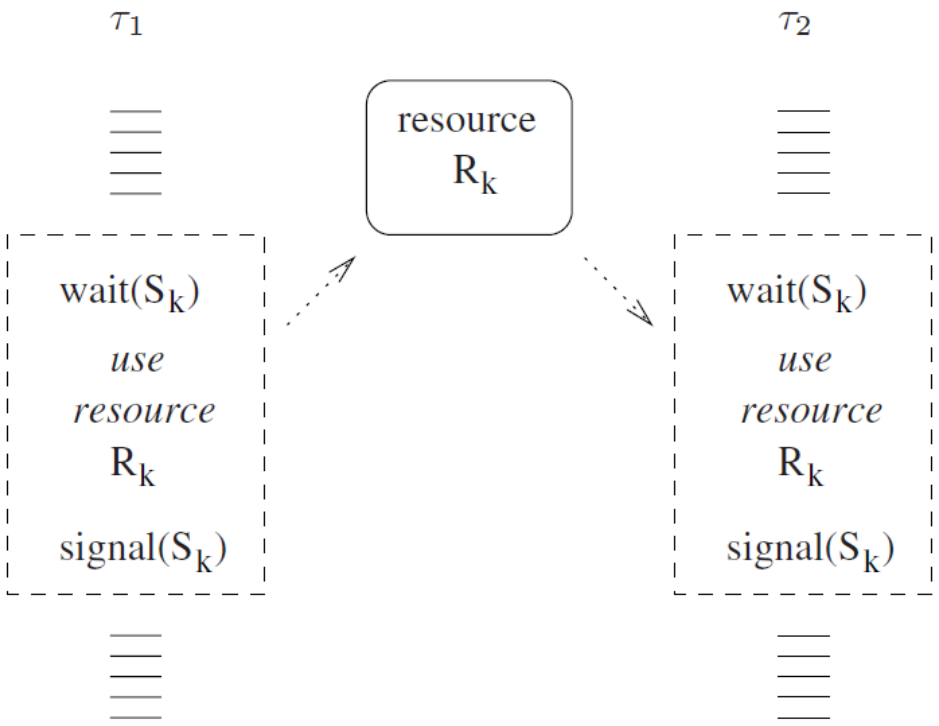
Introduction (4)

- All tasks blocked on a resource are kept in a queue associated with the semaphore that protects the resource.
- When a running task executes a wait primitive on a locked semaphore, it enters a waiting state, until another task executes a signal primitive that unlocks the semaphore.
- When a task leaves the waiting state, it does not go in the running state, but in the ready state, so that the CPU can be assigned to the highest priority task by the scheduling algorithm.



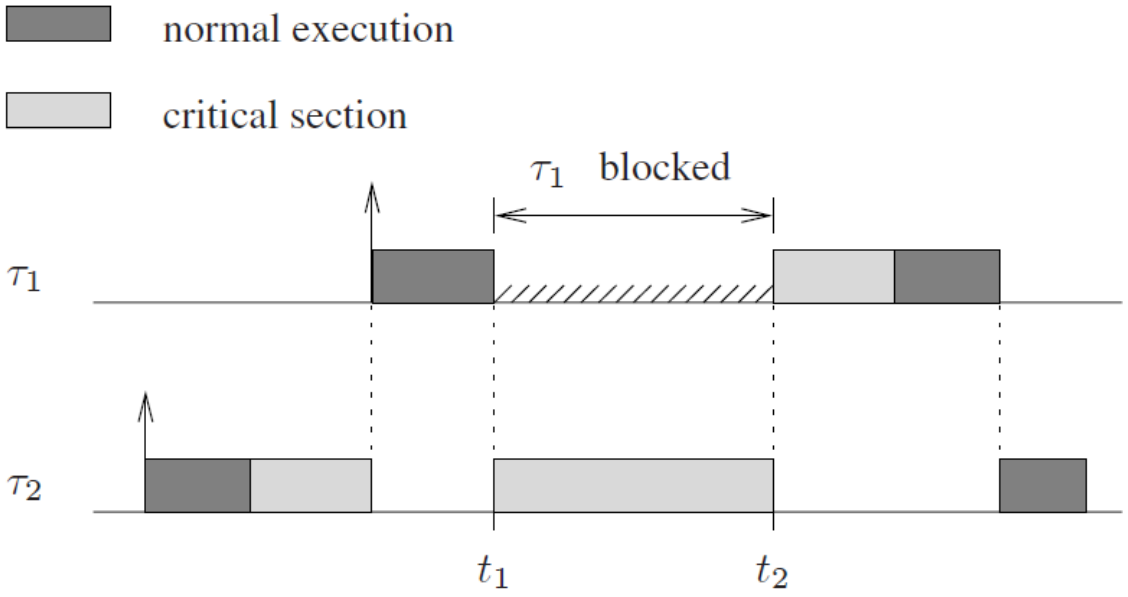
The Priority Inversion Phenomenon

- Consider two tasks τ_1 and τ_2 that share an exclusive resource R_k (such as a list) on which two operations (such as insert and remove) are defined.
- To guarantee the mutual exclusion, both operations must be defined as critical sections.
- If a binary semaphore S_k is used for this purpose, then each critical section must begin with a wait(S_k) primitive and must end with a signal(S_k) primitive.



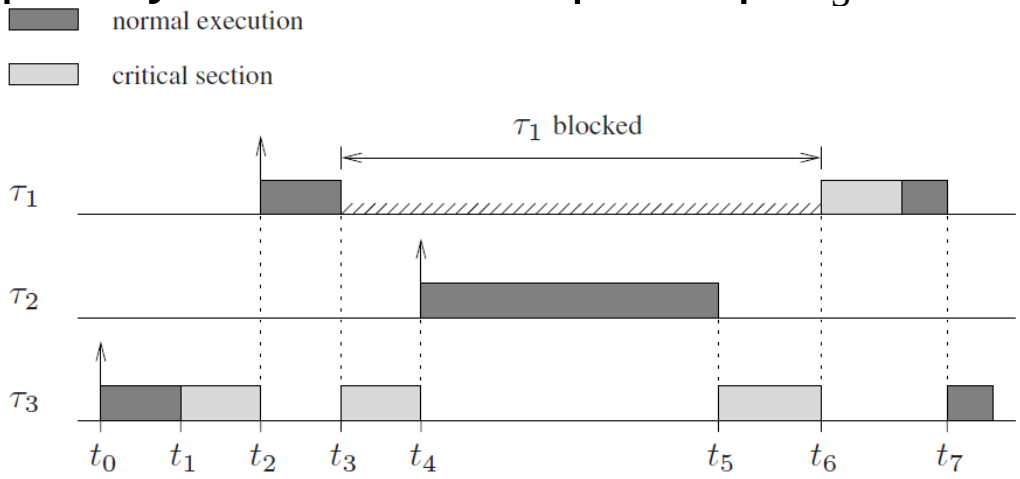
The Priority Inversion Phenomenon (2)

- If preemption is allowed and τ_1 has a higher priority than τ_2, then τ_1 can be blocked.
- The maximum blocking time that τ_1 may experience is equal to the time needed by τ_2 to execute its critical section.



The Priority Inversion Phenomenon (3)

- Unfortunately, the blocking time of a task on a busy resource cannot be bounded by the duration of the critical section executed by the lower-priority task.
- A *priority inversion* is said to occur in the interval $[t_3, t_6]$, since the highest-priority task τ_1 waits for the execution of lower priority tasks (τ_2 and τ_3).
- The duration of priority inversion is unbounded, since any intermediate-priority task that can preempt τ_3 will indirectly block τ_1.



Several Approaches to Avoid Priority Inversion

- The rest of this lecture presents the following resource access protocols:
  1. Non-Preemptive Protocol (NPP);
  2. Highest Locker Priority (HLP), also called Immediate Priority Ceiling (IPC);
  3. Priority Inheritance Protocol (PIP);
  4. Priority Ceiling Protocol (PCP);
  5. Stack Resource Policy (SRP);

Terminology and Assumptions

- Consider a set of n periodic tasks, τ_1, τ_2, ..., τ_n, which cooperate through m shared resources, R_1, R_2, ..., R_m.
- Each task is characterized by a period T_i and a worst-case computation time C_i.
- Each resource R_k is guarded by a distinct semaphore S_k.
- All critical sections on resource R_k begin with a wait(S_k) operation and end with a signal(S_k) operation.
- Since a protocol modifies the task priority, each task is characterized by a fixed nominal priority P_i (assigned, for example, by the Rate Monotonic algorithm) and an active priority p_i ($p_i \geq P_i$), which is dynamic and initially set to P_i.

Terminology and Assumptions (2)

- B_i denotes the maximum blocking time that task τ_i can experience.
- $z_{i,k}$ denotes a generic critical section of task τ_i guarded by semaphore S_k.
- $Z_{i,k}$ denotes the longest critical section of task τ_i guarded by semaphore S_k.
- $\delta_{i,k}$ denotes the duration of $Z_{i,k}$.
- $z_{i,h} \subset z_{i,k}$ indicates that $z_{i,h}$ is entirely contained in $z_{i,k}$.
- σ_i denotes the set of semaphores used by τ_i.
- $\sigma_{i,j}$ denotes the set of semaphores that can block τ_i, used by the lower-priority task τ_j.

Terminology and Assumptions (3)

- $\gamma_{i,j}$ denotes the set of the longest critical sections that can block τ_i, accessed by the lower priority task τ_j. That is

$$\gamma_{i,j} = \{Z_{j,k} \mid (P_j < P_i) \text{ and } (S_k \in \sigma_{i,j})\}$$

- γ_i denotes the set of all the longest critical sections that can block τ_i. That is,

$$\gamma_i = \bigcup_{j:P_j<P_i} \gamma_{i,j}$$

Terminology and Assumptions (4)

- The properties of the protocols are valid under the following assumptions:
  - Tasks $\tau_1, \tau_2, \ldots, \tau_n$ are assumed to have different priorities and are listed in descending order of nominal priority, with τ1 having the highest nominal priority.
  - Tasks do not suspend themselves on I/O operations or on explicit synchronization primitives (except on locked semaphores).
  - The critical sections used by any task are properly nested; that is, given any pair

$$z_{i,h} \text{ and } z_{i,k}, \text{ then either } z_{i,h} \subset z_{i,k}, z_{i,k} \subset z_{i,h}, \text{ or } z_{i,h} \cap z_{i,k} = \emptyset.$$

  - Critical sections are guarded by binary semaphores, meaning that only one task at a time can be within a critical section.

Non-Preemptive Protocol

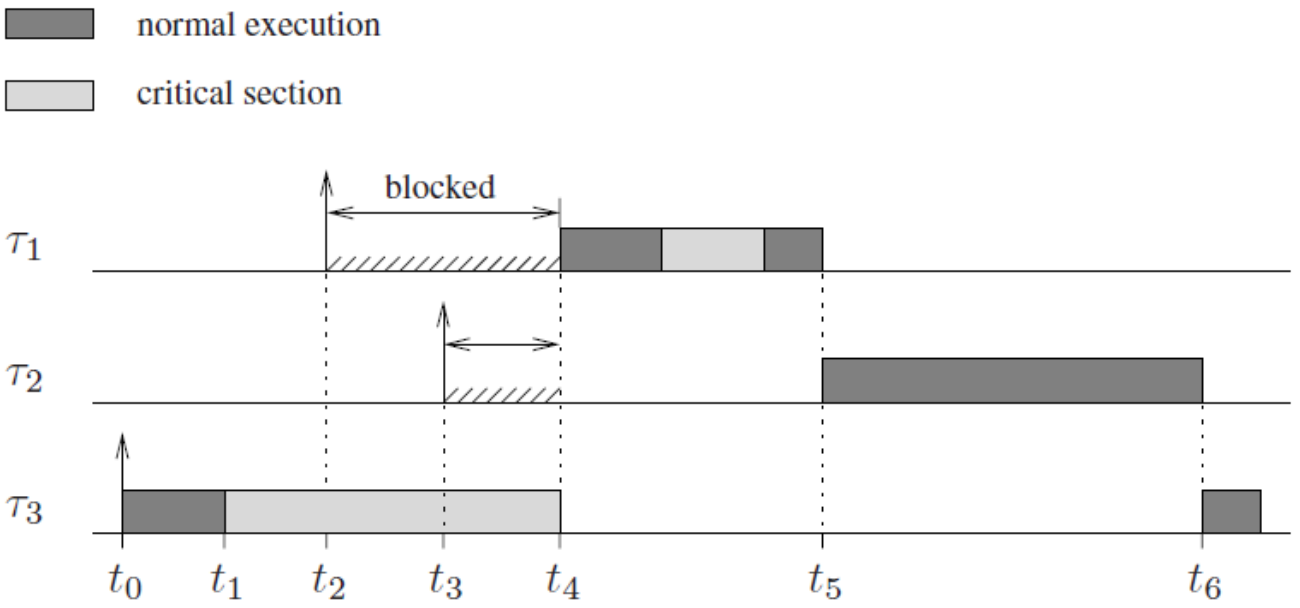
- A simple solution that avoids the unbounded priority inversion problem is to disallow preemption during the execution of any critical section.
- This method, also referred to as *Non-Preemptive Protocol* (NPP), can be implemented by raising the priority of a task to the highest priority level whenever it enters a shared resource.
- In particular, as soon as a task τ_i enters a resource R_k, its dynamic priority is raised to the level:

$$p_i(R_k) = \max_h \{P_h\}.$$

- The dynamic priority is then reset to the nominal value P_i when the task exits the critical section.

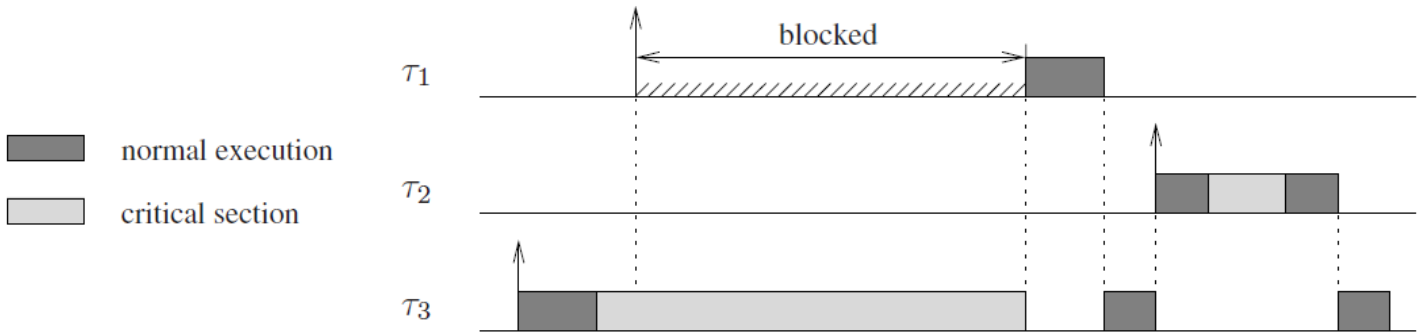
Solving the Priority Inversion Phenomenon through NPP

- How NPP solves the priority inversion phenomenon.



Solving the Priority Inversion Phenomenon through NPP (2)

- NPP is only appropriate when tasks use short critical sections because it creates unnecessary blocking.
- Consider, for example, where τ_1 is the highest-priority task that does not use any resource, whereas τ_2 and τ_3 are low-priority tasks that share an exclusive resource.
- If the low-priority task τ_3 enters a long critical section, τ_1 may unnecessarily be blocked for a long period of time.



Highest Locker Priority Protocol

- The *Highest Locker Priority* (HLP) protocol improves NPP by raising the priority of a task that enters a resource R_k to the highest priority among the tasks sharing that resource.
- In particular, as soon as a task τ_i enters a resource R_k, its dynamic priority is raised to the level:

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

- The dynamic priority is then reset to the nominal value P_i when the task exits the critical section.

Highest Locker Priority Protocol

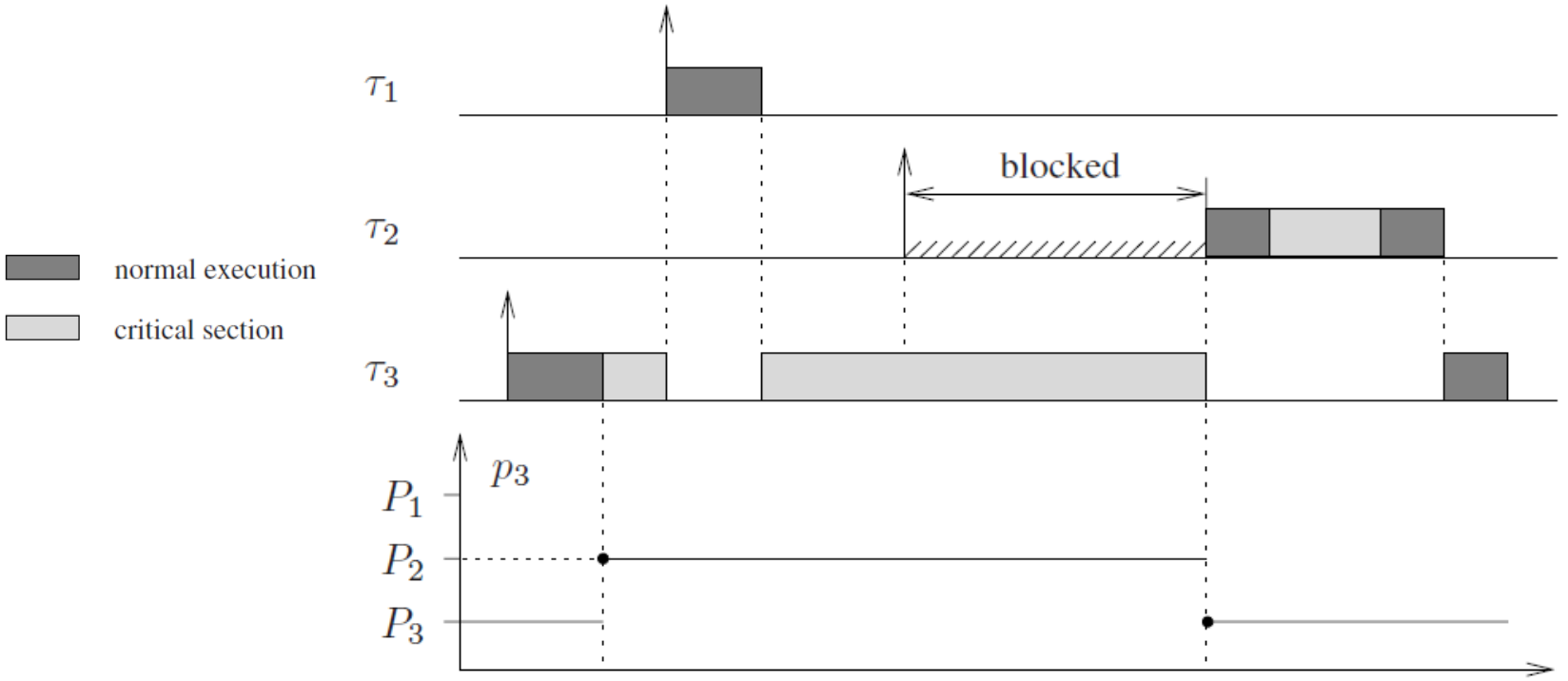
- The online computation of the priority level in the above equation can be simplified by assigning each resource R_k a priority ceiling $C(R_k)$ (computed offline) equal to the maximum priority of the tasks sharing R_k; that is:

$$C(R_k) \stackrel{\text{def}}{=} \max_h \{P_h \mid \tau_h \text{ uses } R_k\}.$$

- Then, as soon as a task τ_i enters a resource R_k, its dynamic priority is raised to the ceiling of the resource.
  - For this reason, this protocol is also referred to as *Immediate Priority Ceiling*.

An Example of Highest Locker Priority Protocol

- The ceiling of the shared resource is P_2;
  - τ_1 can preempt τ_3 inside the critical section, whereas τ_2 is blocked until τ_3 exits its critical section.



Priority Inheritance Protocol

- The Priority Inheritance Protocol (PIP), proposed by Sha, Rajkumar and Lehoczky [SRL90], avoids unbounded priority inversion by modifying the priority of those tasks that cause blocking.
- In particular, when a task τ_i blocks one or more higher-priority tasks, it temporarily assumes (inherits) the highest priority of the blocked tasks.
- This prevents medium-priority tasks from preempting τ_i and prolonging the blocking duration experienced by the higher-priority tasks.

Priority Inheritance Protocol: Protocol Definition

- The Priority Inheritance Protocol can be defined as follows:
  - Tasks are scheduled based on their active priorities. Tasks with the same priority are executed in a First Come First Served discipline.
  - When task τ_i tries to enter a critical section $z_{i,k}$ and resource R_k is already held by a lower-priority task τ_j, then τ_i is blocked. τ_i is said to be blocked by the task τ_j that holds the resource. Otherwise, τ_i enters the critical section $z_{i,k}$.
  - When a task τ_i is blocked on a semaphore, it transmits its active priority to the task, say τ_j, that holds that semaphore. Hence, τ_j resumes and executes the rest of its critical section with a priority $p_j=p_i$. Task τ_j is said to inherit the priority of τ_i. In general, a task inherits the highest priority of the tasks it blocks. That is, at every instant,

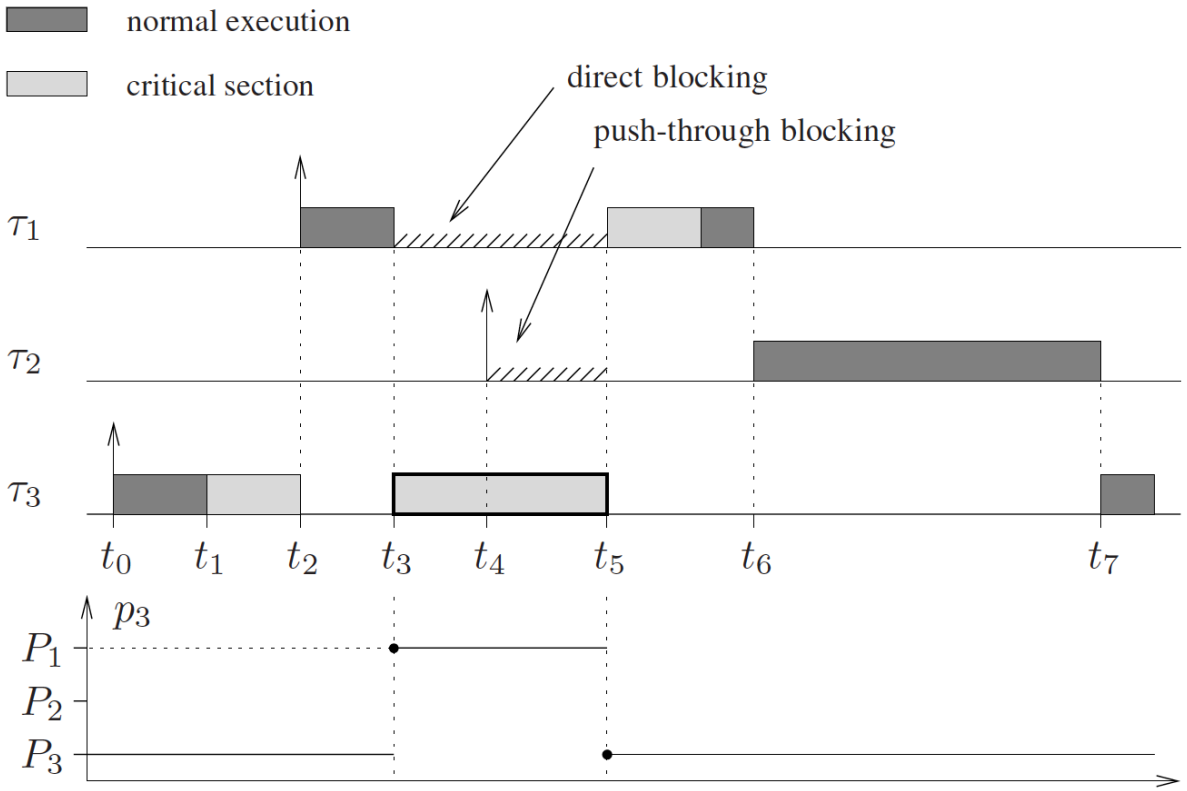
$$p_j(R_k) = \max\{P_j,\ \max_h\{P_h | \tau_h \text{ is blocked on } R_k\}\}.$$

Priority Inheritance Protocol: Protocol Definition (2)

- The Priority Inheritance Protocol can be defined as follows:
  - When τ_j exits a critical section, it unlocks the semaphore, and the highest-priority task blocked on that semaphore, if any, is awakened. Moreover, the active priority of τ_j is updated as follows: if no other tasks are blocked by τ_j, p_j is set to its nominal priority P_j; otherwise it is set to the highest priority of the tasks blocked by τ_j, according to the previous equation.
  - Priority inheritance is transitive; that is, if a task τ_3 blocks a task τ_2, and τ_2 blocks a task τ_1, then τ_3 inherits the priority of τ_1 via τ_2.

Example of Priority Inheritance Protocol

- Showing how the priority inversion phenomenon can be bounded by the Priority Inheritance Protocol.

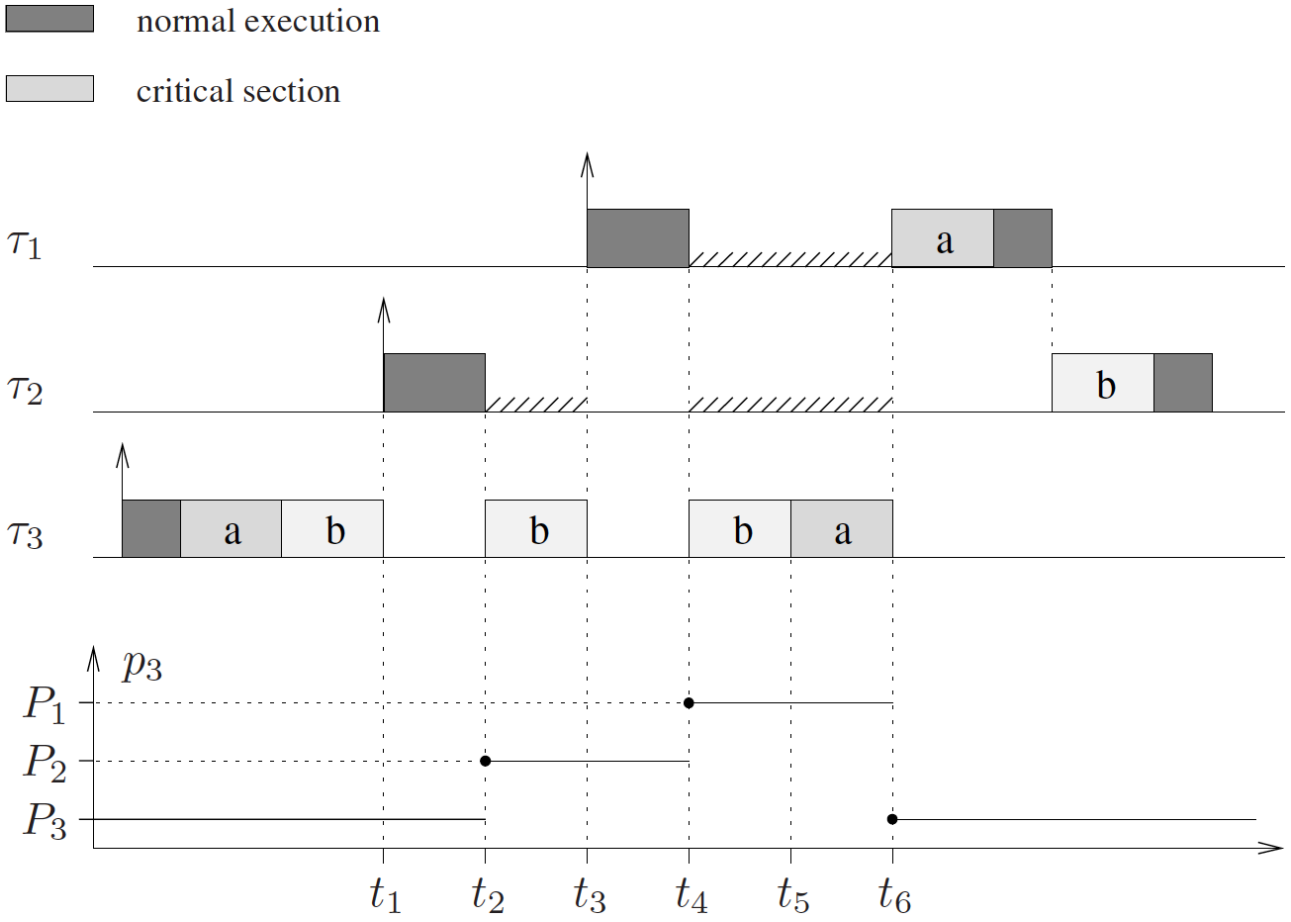


Two kinds of blocking of PIP

- A high-priority task can experience two kinds of blocking
  - **Direct blocking:** It occurs when a higher-priority task tries to acquire a resource already held by a lower-priority task. Direct blocking is necessary to ensure the consistency of the shared resources.
  - **Push-through blocking:** It occurs when a medium-priority task is blocked by a low-priority task that has inherited a higher priority from a task it directly blocks. Push-through blocking is necessary to avoid unbounded priority inversion.

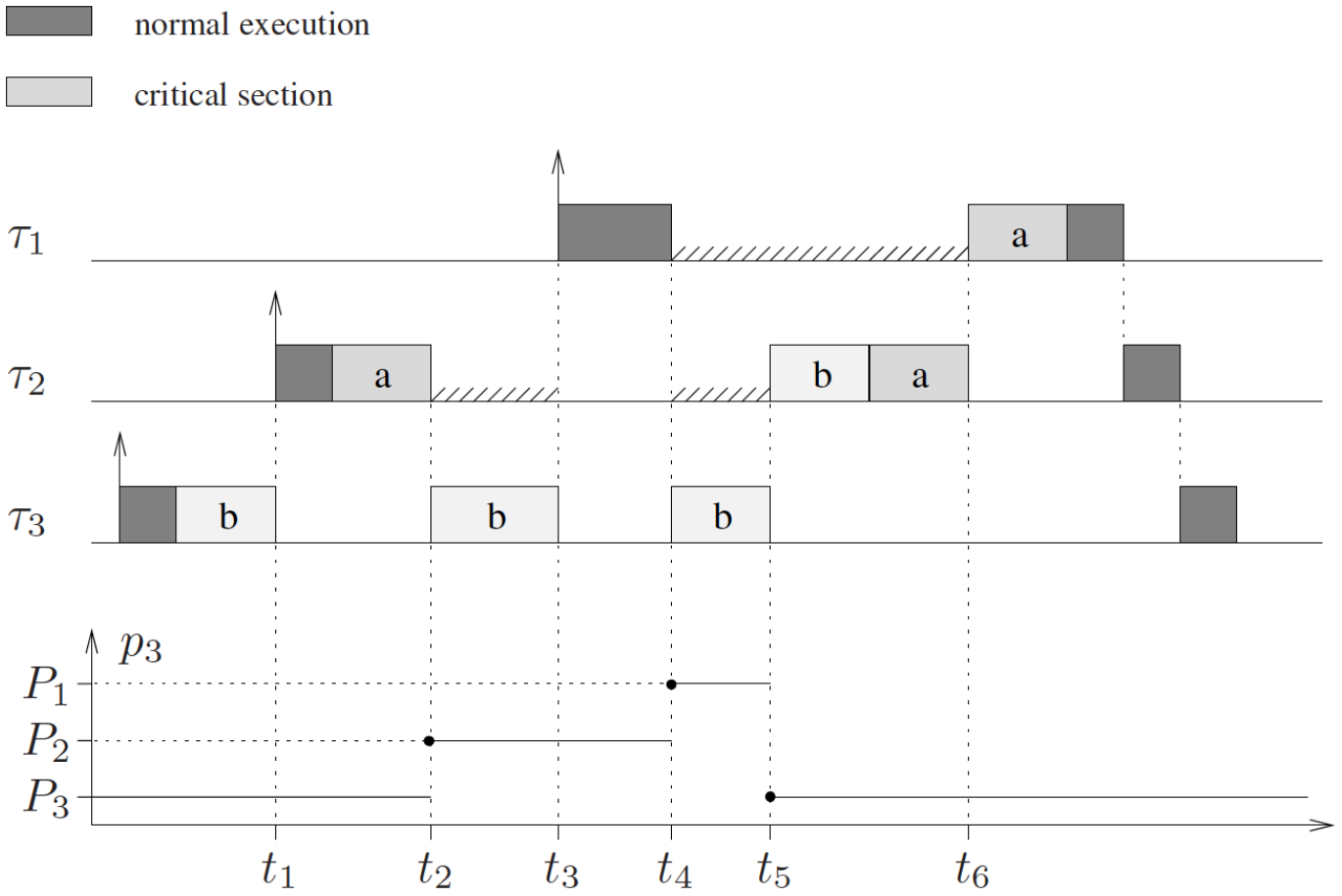
Another Example of Priority Inheritance Protocol

- When a task exits a critical section, it resumes the priority it had when it entered. This, however, is not always true.



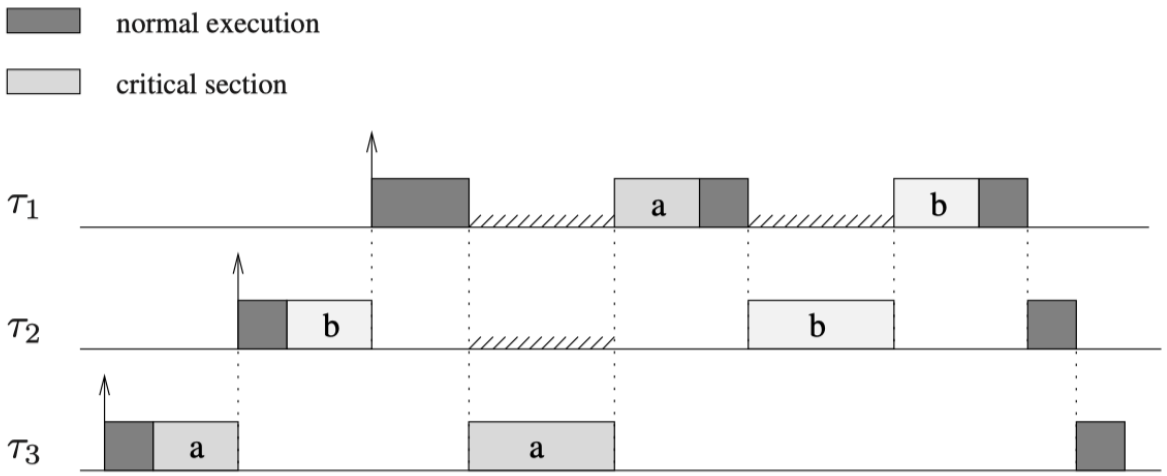
Example of Transitive Priority Inheritance

- An example of transitive priority inheritance is shown in the following figure.



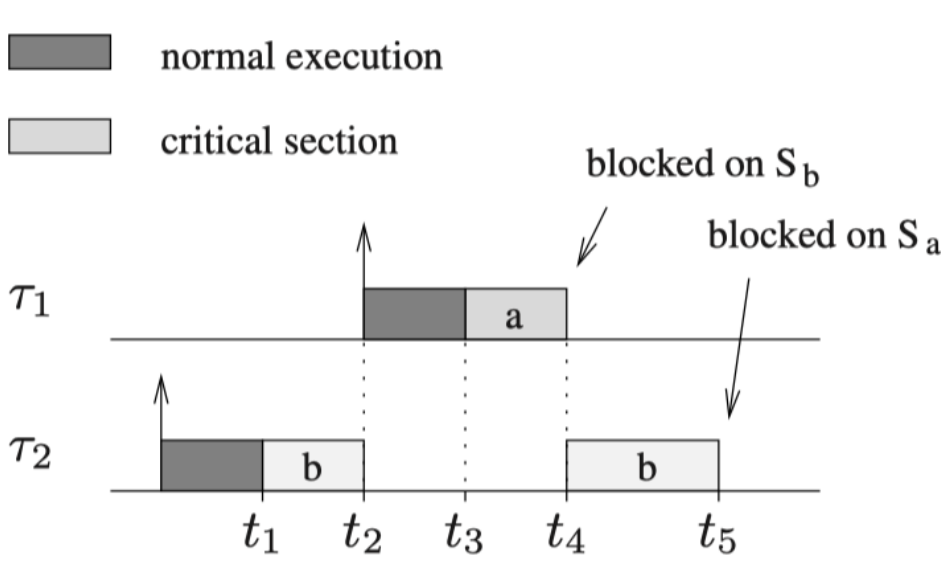
CHAINED BLOCKING

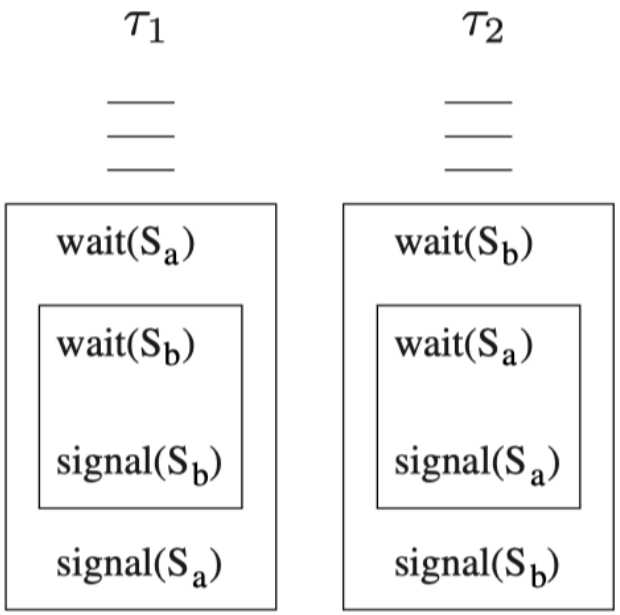
- In the worst case, if τ_1 accesses *n* distinct semaphores that have been locked by n lower-priority tasks, τ_1 will be blocked for the duration of *n* critical sections.



DEADLOCK

- In this case, the deadlock problem can be solved by imposing a total ordering on the semaphore accesses.





Priority Ceiling Protocol

- The *Priority Ceiling Protocol* (PCP) was introduced by Sha, Rajkumar, and Lehoczky [SRL90] to bound the priority inversion phenomenon and prevent the formation of deadlocks and chained blocking.
- The basic idea of this method is to extend the Priority Inheritance Protocol with a rule for granting a lock request on a free semaphore.
- To avoid multiple blocking, this rule does not allow a task to enter a critical section if there are locked semaphores that could block it.
  - This means that once a task enters its first critical section, it can never be blocked by lower-priority tasks until its completion.

Priority Ceiling Protocol (2)

- In order to realize this idea, each semaphore is assigned a *priority ceiling* equal to the highest priority of the tasks that can lock it.
- A task τ_i is allowed to enter a critical section only if its priority is higher than all priority ceilings of the semaphores currently locked by tasks other than τ_i.

Definition of Priority Ceiling Protocol

- Each semaphore S_k is assigned a priority ceiling $C(S_k)$ equal to the highest priority of the tasks that can lock it. Note that $C(S_k)$ is a static value that can be computed off-line:

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

- Let τ_i be the task with the highest priority among all tasks ready to run; thus, τ_i is assigned the processor.
- Let S^* be the semaphore with the highest ceiling among all the semaphores currently locked by tasks other than τ_i and let $C(S^*)$ be its ceiling.
- To enter a critical section guarded by a semaphore S_k, τ_i must have a priority higher than $C(S^*)$. If $P_i \leq C(S^*)$, the lock on S_k is denied and τ_i is said to be blocked on semaphore S^* by the task that holds the lock on S^*.

Definition of Priority Ceiling Protocol (2)

- When a task τ_i is blocked on a semaphore, it transmits its priority to the task, say τ_j, that holds that semaphore. Hence, τ_j resumes and executes the rest of its critical section with the priority of τ_i. Task τ_j is said to inherit the priority of τ_i.
- In general, a task inherits the highest priority of the tasks blocked by it. That is, at every instant,

$$p_j(R_k) = \max\{P_j,\ \max_h\{P_h | \tau_h \text{ is blocked on } R_k\}\}.$$

- When τ_j exits a critical section, it unlocks the semaphore and the highest-priority task, if any, blocked on that semaphore is awakened. Moreover, the active priority of τ_j is updated as follows:
  - If no other tasks are blocked by τ_j, p_j is set to the nominal priority P_j; otherwise, it is set to the highest priority of the tasks blocked by τ_j, according to the above equation.

Definition of Priority Ceiling Protocol (3)

- Priority inheritance is transitive; that is, if a task τ_3 blocks a task τ_2, and τ_2 blocks a task τ_1, then τ_3 inherits the priority of τ_1 via τ_2.

Example of Priority Ceiling Protocol

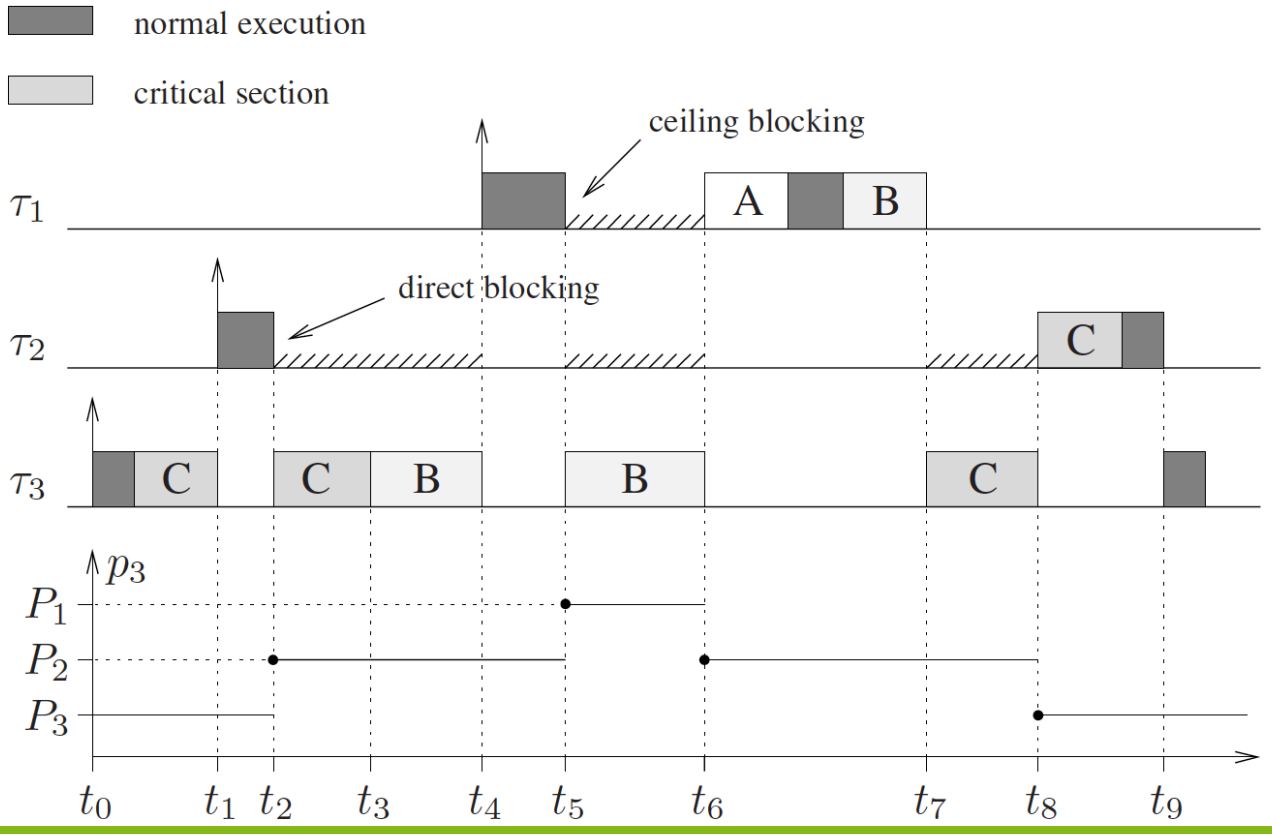
- Consider three tasks τ_1, τ_2, and τ_3 having decreasing priorities.
  - The highest-priority task τ_1 sequentially accesses two critical sections guarded by semaphores S_A and S_B;
  - Task τ_2 accesses only a critical section guarded by semaphore S_C;
  - Task τ_3 uses semaphore S_C and then makes a nested access to S_B.
  - From tasks' resource requirements, all semaphores are assigned the following priority ceilings:

$$\begin{cases} C(S_A) = P_1 \\ C(S_B) = P_1 \\ C(S_C) = P_2. \end{cases}$$

Example of Priority Ceiling Protocol (2)

- Note that the Priority Ceiling Protocol introduces a third form of blocking, called *ceiling blocking*.
- A ceiling blocking is experienced by task τ_1 at time t_5.

$$\begin{cases} C(S_A) = P_1 \\ C(S_B) = P_1 \\ C(S_C) = P_2. \end{cases}$$



Stack Resource Policy

- The *Stack Resource Policy* (SRP) is a technique proposed by Baker [Bak91] for accessing shared resources.
- It extends the Priority Ceiling Protocol (PCP) in three essential points:
  1. It allows the use of multi-unit resources.
  2. It supports dynamic priority scheduling.
  3. It allows the sharing of runtime stack-based resources.

Stack Resource Policy (2)

- From a scheduling point of view, the essential difference between the PCP and the SRP is on the time at which a task is blocked.
  - Whereas under the PCP a task is blocked at the time it makes its first resource request, under the SRP a task is blocked at the time it attempts to preempt.
  - This early blocking slightly reduces concurrency but saves unnecessary context switches, simplifies the implementation of the protocol, and allows the sharing of runtime stack resources.

Some Definitions

- **Priority:**
  - Each task τ_i is assigned a priority p_i that indicates the importance (that is, the urgency) of τ_i with respect to the other tasks in the system.
  - Priorities can be assigned to tasks either statically or dynamically.
  - At any time t, $p_a > p_b$ means that the execution of τ_a is more important than that of τ_b; hence, τ_b can be delayed in favor of τ_a.
  - For example, priorities can be assigned to tasks based on Rate Monotonic (RM) or Earliest Deadline First (EDF).

Some Definitions (2)

- **Preemption Level:**
  - Besides a priority p_i, a task τ_i is characterized by a *preemption level* π_i.
    - The preemption level is a static parameter, assigned to a task at its creation time and associated with all instances of that task.
  - The essential property of preemption levels is that a task τ_a can preempt another task τ_b only if $\pi_a > \pi_b$. This is also true for priorities.
  - The reason for distinguishing preemption levels from priorities is that preemption levels are fixed values that can be used to predict potential blocking also in the presence of dynamic priority schemes.
  - The general definition of preemption level used to prove all properties of the SRP requires that:
    - If τ_a arrives after τ_b and τ_a has higher priority than τ_b, then τ_a must have a higher preemption level than τ_b.

Some Definitions (4)

- **Resource Units:**
  - Each resource R_k is allowed to have N_k units that can be concurrently accessed by multiple tasks. This notion generalizes the classical single-unit resource, which can be accessed by one task at the time under mutual exclusion.
  - A resource with N_k units can be concurrently accessed by N_k different tasks, if each task requires one unit.
  - In general, n_k denotes the number of currently available units for R_k, meaning that $N_k - n_k$ units are locked. If $n_k = 0$, a task requiring 3 units of R_k is blocked until n_k becomes greater than or equal to 3.

Some Definitions (5)

- **Resource Requirements:**
  - When entering a critical section $z_{i,k}$ guarded by a multi-unit semaphore S_k, a task τ_i must specify the number of units it needs by calling a $wait(S_k, r)$, where r is the number of requested units.
  - When exiting the critical section, τ_i must call a $signal(S_k)$, which releases all the r units.
  - The maximum number of units that can be simultaneously requested by τ_i to R_k is denoted by $\mu_i(R_k)$, which can be derived off-line by analyzing the task code.
  - It is clear that if τ_i requires more units than are available, that is, if $\mu_i(R_k)>n_k$, then τ_i blocks until n_k becomes greater than or equal to $\mu_i(R_k)$.

Some Definitions (6)

- **Resource Ceiling:**
  - Each resource R_k (or semaphore S_k) is assigned a *ceiling* $C_{Rk}(n_k)$ equal to the highest preemption level of the tasks that could be blocked on R_k if issuing their maximum request.
  - Hence, the ceiling of a resource is a dynamic value, which is a function of the units of R_k that are currently available. That is,

$$C_{R_k}(n_k) = \max\{\pi_i \mid \mu_i(R_k) > n_k\}.$$

  - If all units of R_k are available, that is, if n_k=N_k, then $CR_k(N_k)$= 0.

An Example of Resource Ceiling

- Consider three tasks (τ_1, τ_2, τ_3) share three resources (R_1, R_2, R_3), consisting of three, one, and three units, respectively.

	D_i	π_i	$\mu_i(R_1)$	$\mu_i(R_2)$	$\mu_i(R_3)$
τ_1	5	3	1	0	1
τ_2	10	2	2	1	3
τ_3	20	1	3	1	1

- Based on these requirements, the current ceilings of the resources as a function of the number n_k of available units are:

	$C_R(3)$	$C_R(2)$	$C_R(1)$	$C_R(0)$
R_1	0	1	2	3
R_2	-	-	0	2
R_3	0	2	2	3

Some Definitions (7)

- **System Ceiling:**
  - The resource access protocol adopted in the SRP also requires a *system ceiling*, Π_s, defined as the maximum of the current ceilings of all the resources; that is,

$$\Pi_s = \max_k \{C_{R_k}\}.$$

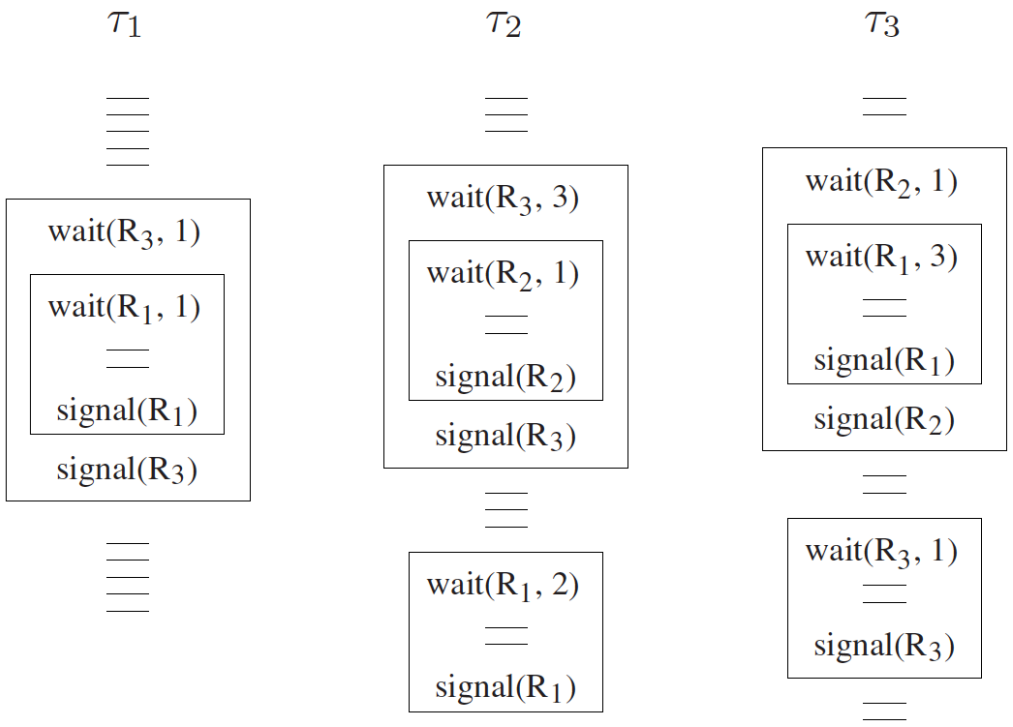
  - Notice that Π_s is a dynamic parameter that can change every time a resource is accessed or released by a task.

Protocol Definition of SRP

- The key idea of the SRP is that when a task needs a resource that is not available, it is blocked at the time it attempts to preempt rather than later.
- In order to prevent multiple priority inversions, a task is not allowed to start until the resources currently available are sufficient to meet the maximum requirement of every task that could preempt it.
- This is achieved by the following preemption test:
  - **SRP Preemption Test:** A task is not permitted to preempt until its priority is the highest among those of all the tasks ready to run, and its preemption level is higher than the system ceiling.

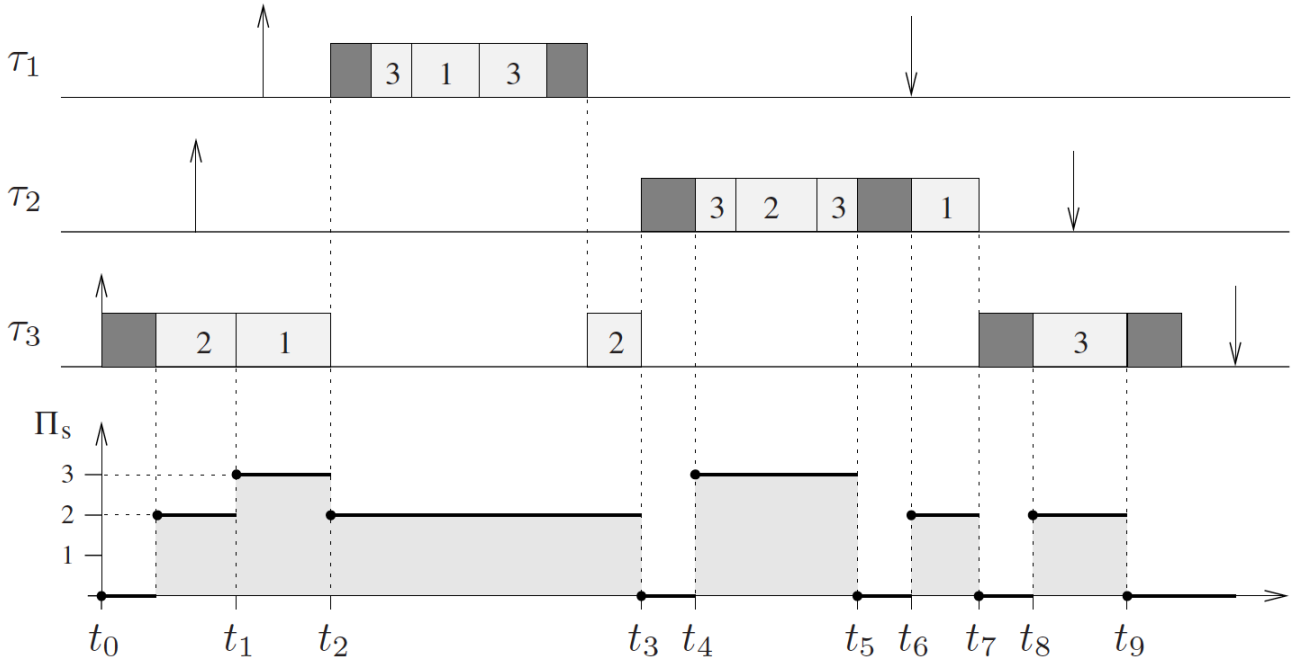
An Example of Stack Resource Policy

- In order to illustrate how the SRP works, consider the task set already described in Slide 43.
- *wait*(R_k, r) denotes the request of r units of resource R_k, and *signal*(R_k) denotes their release.



An Example of Stack Resource Policy (2)

- A possible EDF schedule for the task set is depicted in the following figure.



Evaluation Summary of Resource Access Protocols

	priority	Num. of blocking	pessimism	blocking instant	transpa-rency	deadlock preven-tion	implem-entation
NPP	any	1	high	on arrival	YES	YES	easy
HLP	fixed	1	medium	on arrival	NO	YES	easy
PIP	fixed	α_i	low	on access	YES	NO	hard
PCP	fixed	1	medium	on access	NO	YES	medium
SRP	any	1	medium	on arrival	NO	YES	easy

Summary of Resource Access Protocols

- Introduction to Resource Access Protocols
- The Priority Inversion Phenomenon
- Terminology and Assumptions
- Resource access protocols:
  - Non-Preemptive Protocol (NPP);
  - Highest Locker Priority (HLP), also called Immediate Priority Ceiling (IPC);
  - Priority Inheritance Protocol (PIP);
  - Priority Ceiling Protocol (PCP);
  - Stack Resource Policy (SRP);