Resource Access Protocols

IL2212 Embedded Software

Ingo Sander

KTH Royal Institute of Technology
Stockholm, Sweden
ingo@kth.se

Outline

Outline

Resource Access Protocols

- Previous discussion assumed independent tasks
- In practice
  - tasks communicate via communication objects like semaphores, message queues or protected objects with each other
  - $\Rightarrow$ tasks are not independent

Resources

Definition: Resource (Buttazzo, 2011)

- A resource S is any software structure that can be used by a task to advance its execution[a].
- A resource
  - that is dedicated to a particular task is called a private resource;
  - that can be used by more tasks is called a shared resource;
  - that is protected against concurrent access is called a exclusive resource.

[a] The symbol S is used, because the symbol R is already occupied to denote the response time. Also, Buttazzo (2011) uses the semaphore as example for a resource.

- Resources can have several units, e.g. counting semaphore
- Following discussion only deals with exclusive resources with one unit
- Exclusive resources are allocated to tasks on a non-preemptive basis and used in a mutually exclusive manner

Mutually Exclusive Resource Access

Assumptions

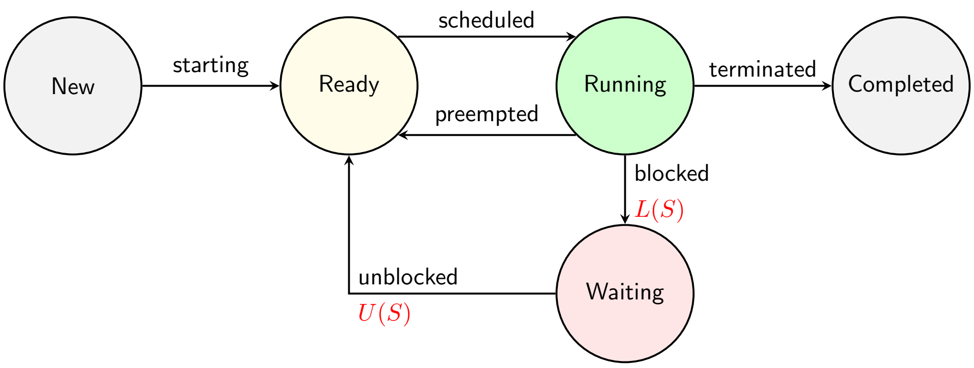
- A lock-based concurrency control mechanism assumed to be used to enforce mutual exclusive access to resources
- When a task wants to use a resource S_i, it executes a command `lock` $L(S_i)$ to request the resource.
- When a task no longer needs the resource S_i, it releases the resource by executing a command unlock signal $U(S_i)$.

Correspondence to critical sections protected by a semaphore

A critical section protected by a semaphore `S` is modelled as an exclusive resource S

- The command `wait(S)` is modelled by a `lock` command $L(S)$
- The command `signal(S)` is modelled by an `unlock` command $U(S)$

Mutually Exclusive Resource Access



- When a lock request $L(S)$ fails, the requesting job is blocked and loses the processor
- An unlock command $U(S)$ frees the resource and might cause that another task is unblocked
- A task stays blocked until the scheduler grants the resources the task is waiting for

Critical Section

- A segment of a task τ_i that begins with a lock $L(S_k)$ and ends with a matching unlock $U(S_k)$ is called a critical section, and is denoted by $z_{i,k}$.
- The longest critical section of τ_i to a resource S_k is denoted by $Z_{i,k}$.
- The duration of $Z_{i,k}$ is denoted by $\delta_{i,k}$

```
...
-- Critical section
lock(S_k)
-- Access to exclusive resource S_k
unlock(S_k)
-- End of critical section
...
```

Nested Critical Sections

- In nested critical sections, resources are always released in last-in-first-out order ⇒ properly nested critical sections
- $z_{i,h} \subset z_{i,k}$ indicates that the critical section $z_{i,h}$ is entirely contained in $z_{i,k}$.
- A critical section that is not included in other critical sections is called an outermost critical section
- Assumption: All nested critical sections are properly nested

Properly Nested Critical Section

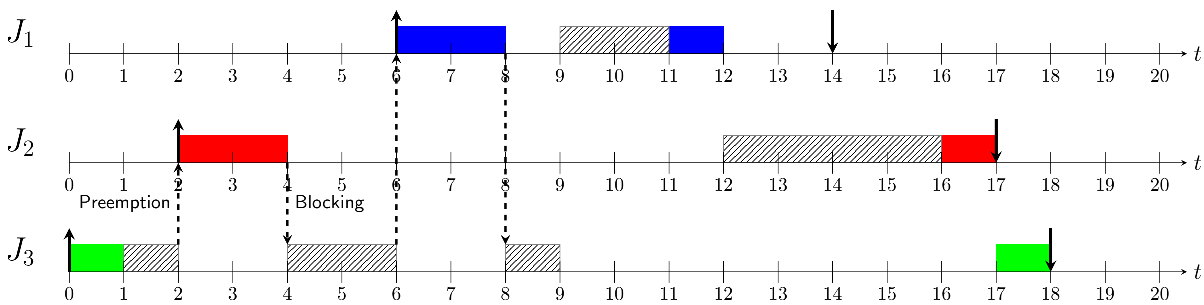
```
lock(S_1)
...
lock(S_2)
..
unlock(S_2)
...
unlock(S_1)
```

Not Properly Nested Critical Section

```
lock(S_2)
...
lock(S_1)
..
unlock(S_2)
...
unlock(S_1)
```

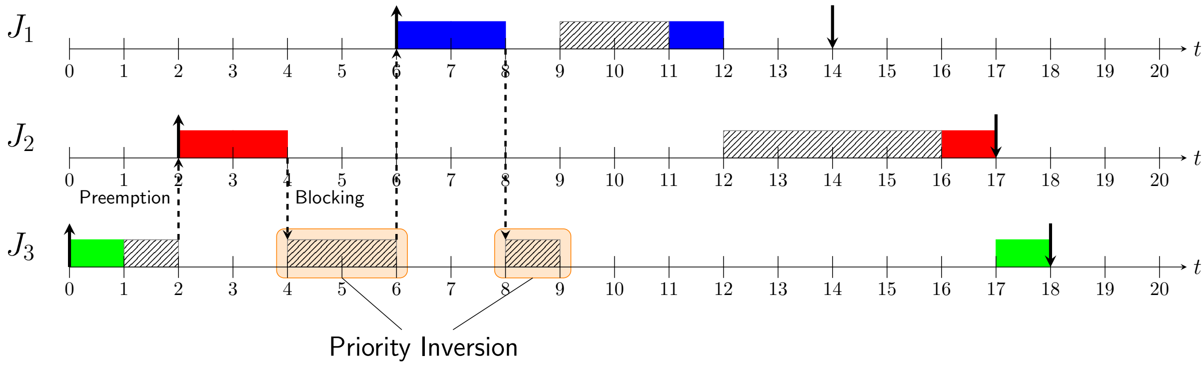

Priority Inversion due to Resource Conflicts

- Jobs J_1, J_2, J_3 have feasible intervals (6,14], (2,17], (0,18].
- Jobs J_1, J_2, J_3 are scheduled using EDF algorithm
- Jobs access critical sections with exclusive resource S_k for the following time units: $\delta_{1,k} = 2$, $\delta_{2,k} = 4$, $\delta_{3,k} = 4$.



Priority Inversion due to Resource Conflicts

- Priority inversion occurs, when a low-priority job executes while a ready higher-priority job waits

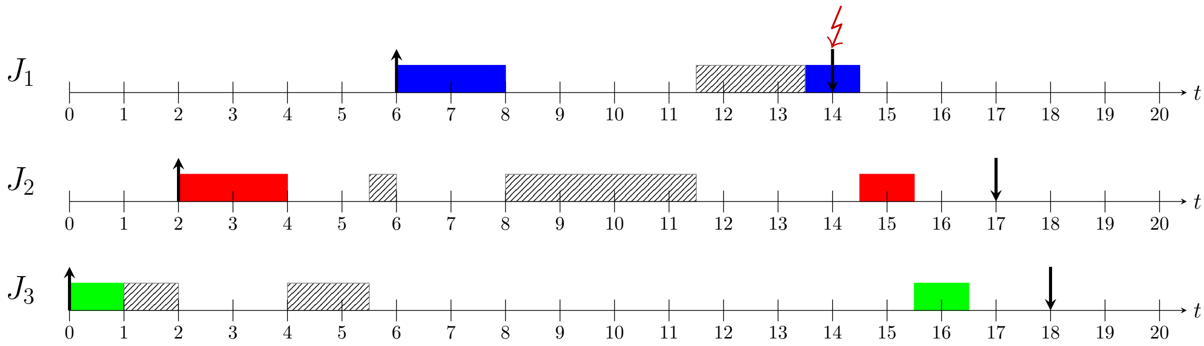


Timing Anomalies

- Timing Anomalies can occur due to priority inversion
- Assume that critical section of job J_3 is reduced to $\delta_{3,k} = 2.5$ instead of $\delta_{3,k} = 4$.

Timing Anomalies

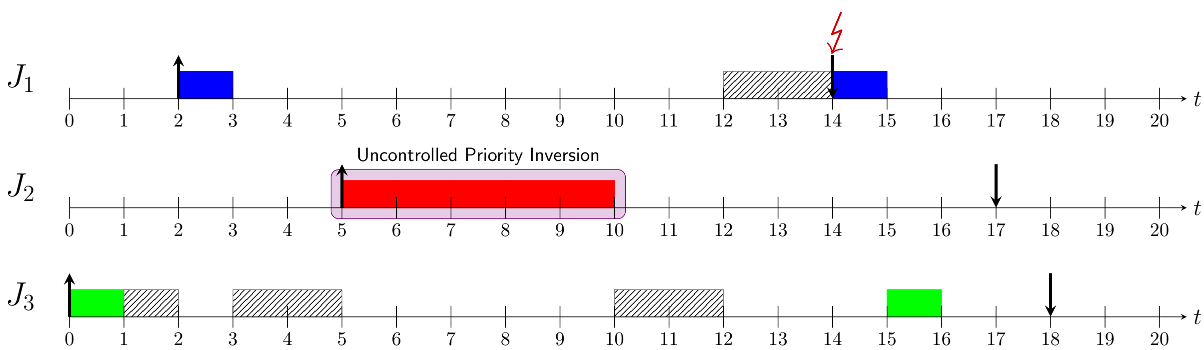
- Timing Anomalies can occur due to priority inversion
- Assume that critical section of job J_3 is reduced to $\delta_{3,k} = 2.5$ instead of $\delta_{3,k} = 4$.



Job J_1 misses its deadline!

Uncontrolled Priority Inversion

- Without suitable protocols, the duration of priority inversion can become unbounded.
- A job J_1 who wants to execute its critical section $z_{1,k}$ can be blocked by a job J_n that holds the resource S_k.
- The duration of of $\delta_{i,k}$ can be prolonged by execution of several jobs J_j with a lower priority than the blocked job and a higher priority than the blocking job.
- This phenomenon is called uncontrolled priority inversion.

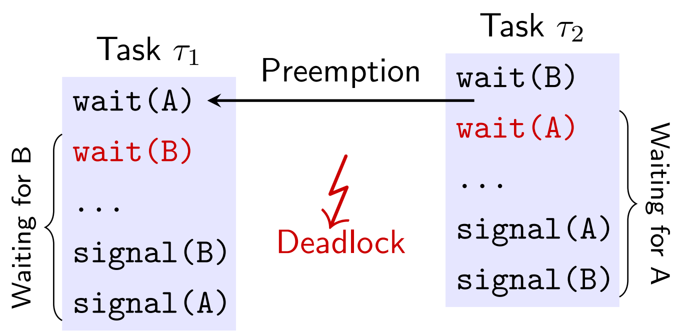


Protocols for Resource Access Control

- Protocols are needed to handle priority inversion in a controlled way and to avoid deadlock

Deadlock

- Deadlock can occur, if tasks block each other from execution

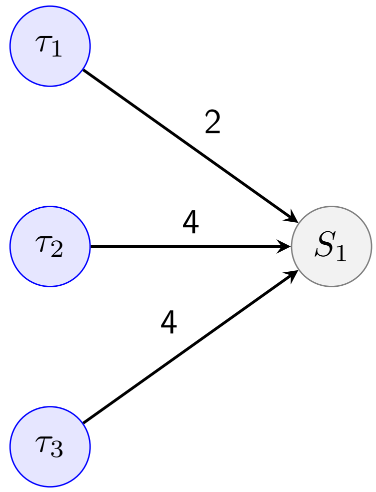


Disciplined approach required to avoid deadlocks

- If a deadlock occurs the program cannot proceed!
- Difficult to test for deadlock, because deadlock situation is difficult to create.

Specification of Resource Requirements

- τ_1 needs S_1 for at most 2 time units $\Rightarrow \delta_{1,1} = 2$
- τ_2 needs S_1 for at most 4 time units $\Rightarrow \delta_{2,1} = 4$
- τ_3 needs S_1 for at most 4 time units $\Rightarrow \delta_{3,1} = 4$



Terminology for Resource Access Protocols

- The maximum blocking time that task τ_i can experience is denoted by B_i.
- The set of the resources a task τ_i uses is denoted by σ_i.
- The set of resources used by the lower priority task j that can block the task τ_i is denoted by $\sigma_{i,j}$.
- $\gamma_{i,j}$ denotes the set of the longest critical sections of task τ_j that can block task τ_i by accessing a resource S_k.

$$\gamma_{i,j} = \{Z_{j,k} | P_j < P_i \wedge S_k \in \sigma_{i,j}\}$$

- γ_i denotes the set of all longest critical sections that can block task τ_i

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

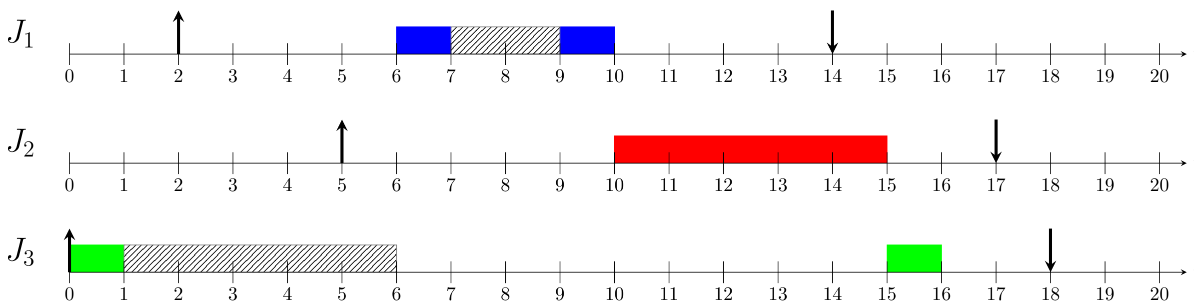
Non-preemptive Protocol (NPP)

- Simplest resource access control protocol
- All critical sections are scheduled non-preemptively, i.e. the current priority $p_i(S_k)$ of a task τ_i that starts executing a critical section $z_{i,k}$ is raised to the highest priority level of all tasks.

$$p_i(S_k) = \max_h \{P_h\}$$

- The current priority of the task τ_i is reset to its assigned priority, when the tasks leaves its critical section.

Non-preemptive Protocol (NPP)



All jobs meet their deadline!

Non-preemptive Protocol (NPP)

- Simple to implement
- Uncontrolled priority inversion cannot occur
- A high-priority job can only be blocked once because of a low-priority job
- Good protocol when critical sections are short
- Blocking time can be calculated

$$B_i = \max_{j,k}\{\delta_{j,k} | Z_{j,k} \in \gamma_i\}$$

where

$$\gamma_{i,j} = \{Z_{j,k} | P_j < P_i, k = 1, \ldots, m\}$$

Resource Access Protocols Non-Preemptive Protocol

Non-preemptive Protocol (NPP)

- Simple to implement
- Uncontrolled priority inversion cannot occur
- A high-priority job can only be blocked once because of a low-priority job
- Good protocol when critical sections are short
- Blocking time can be calculated

$$B_i = \max_{j,k}\{\delta_{j,k} | Z_{j,k} \in \gamma_i\}$$

where

$$\gamma_{i,j} = \{Z_{j,k} | P_j < P_i, k = 1, \ldots, m\}$$

- Every task can be blocked by every lower-priority task even if there is no resource conflict between them

Ingo Sander (KTH) Resource Access Protocols 20 / 40

Priority Inheritance Protocol

- Idea
  - The blocking jobs J_l inherits the priority of the blocked job J_h, when it blocks job J_h
  - Avoids that a high priority job J_h is unnecessarily blocked by a low priority job J_l that does use a different resource than the low priority job J_l.

Priority Inheritance Protocol: Definition (Liu, 2000)

- Definitions
  - The priority of a job J_i according to the scheduling algorithm is its assigned priority.
  - At any time t, each ready job J_i is scheduled and executes at its current priority $p_i(t)$, which may differ from its assigned priority and vary with time.
- Rules
  1. Scheduling Rule
  2. Allocation Rule
  3. Priority Inheritance Rule

Scheduling Rule

- Ready jobs are scheduled on the processor preemptively in a priority-driven manner according to their current priorities.
- At its release time t, the current priority $p_i(t)$ of every job J_i is equal to its assigned priority.
- The job remains at this priority except under the condition stated in the priority-inheritance rule.

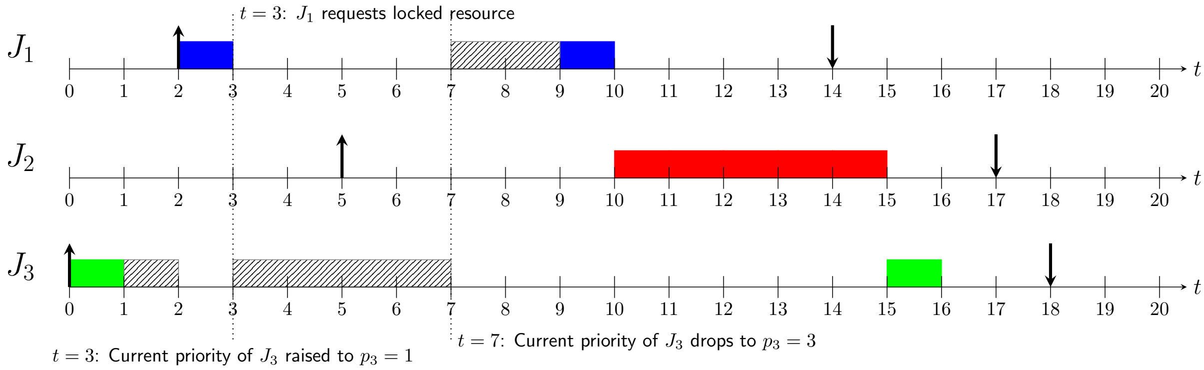
Allocation Rule

- When a job J_i requests a resource S at time t,
  - if S is free, S is allocated to J_i until J releases the resource;
  - if S is not free, the request is denied and J_i is blocked.

Priority Inheritance Rule

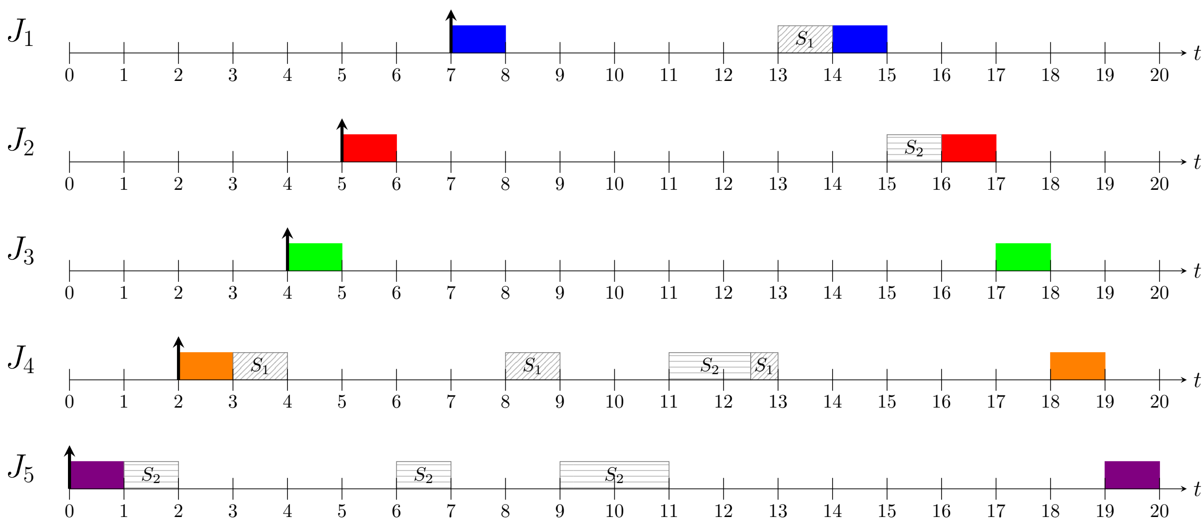
- Priority Inheritance
  - When the requesting job J_i becomes blocked, the job J_l which blocks J_i inherits the current priority $p_i(t)$ of J_i.
  - The job J_l executes at its inherited current priority $p_i(t)$ until it releases the resource S.
  - At that time, the priority of J_l returns to its priority $p_l(t')$ when it acquired the resource S.

Priority Inheritance Protocol



All jobs meet their deadline!

Priority Inheritance Protocol



Priority Inheritance Protocol

- Simple protocol
- Uncontrolled priority inversion cannot occur

Priority Inheritance Protocol

- 👍 Simple protocol
- 👍 Uncontrolled priority inversion cannot occur
- 👎 Protocol does not minimise the blocking time
  - A high priority job J_h can be blocked by several lower priority jobs, if it runs a nested critical section
  - Difficult to calculate blocking times
- 👎 Protocol does not prevent deadlock. External mechanisms are needed.

Priority Ceiling Protocol

- Extends priority inheritance protocol to prevent deadlocks and aims to further reduce the blocking time
- Key Assumptions
  - Assigned priorities of all jobs are fixed
  - Resources required by all jobs are known a priori before the execution of any job begins

Priority Ceiling Protocol: Definition (Liu, 2000)

- Definitions
  - The priority ceiling of any resource S_k is the highest priority of all jobs that require S_k and is denoted $C(S_k)$.
  - At any time t, the current priority ceiling $C(S^*)(t)$ of the system is
    - equal to the highest priority ceiling of the resources that are in use at the time, if some resources are in use
    - otherwise it is Ω, a non-existing priority that is lower than the priority of any job
- Rules
  1. Scheduling Rule
  2. Allocation Rule
  3. Priority Inheritance Rule

Scheduling Rule

- At its release time t, the current priority $p_i(t)$ of every job J_i is equal to its assigned priority.
- The job remains at this priority except under the condition stated in the priority-inheritance rule.
- Every job J_i is scheduled preemptively and in a priority-driven manner at its current priority $p_i(t)$.

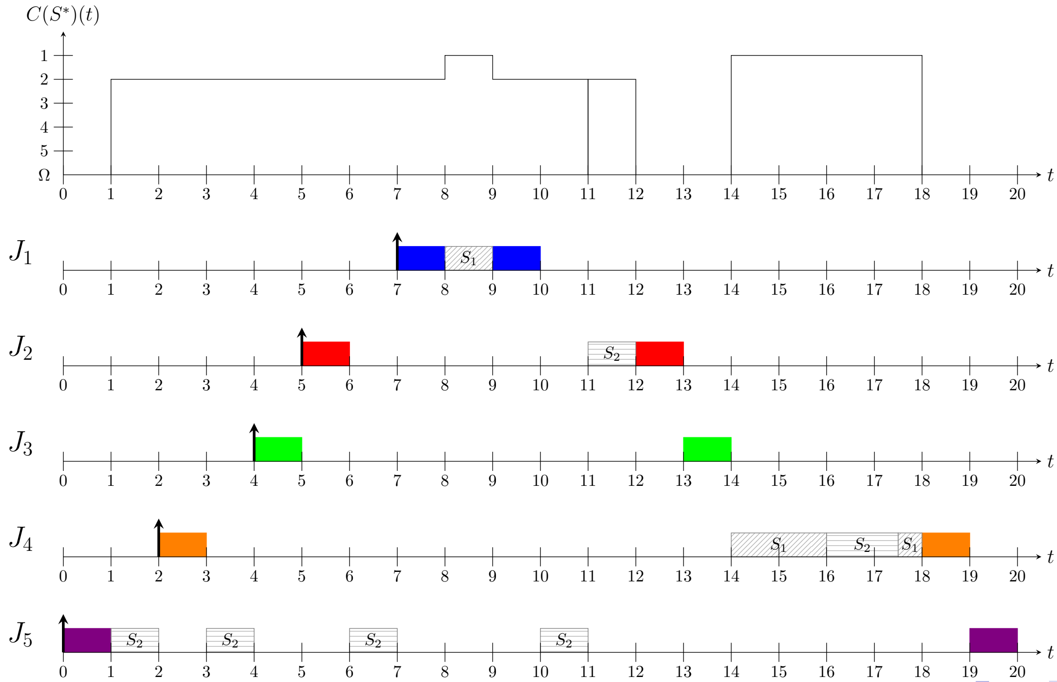
Allocation Rule

- When a job J_i requests a resource S_k at time t,
  1. if S_k is not free, the request is denied and J_i is blocked.
  2. if S_k is free,
     - if J_i's current priority $p_i(t)$ is higher than the current priority ceiling of the system $C(S^*)(t)$, S_k is allocated to J.
     - if J_i's current priority $p_i(t)$ is not higher than the current ceiling of the system $C(S^*)(t)$, S_k is allocated to J_i only if J_i is the job holding the resource(s) whose priority ceiling is equal to $C(S^*)(t)$; otherwise J_i's request is denied and J_i becomes blocked.

Priority Inheritance Rule

- Priority Inheritance
  - When J_i becomes blocked, the job J_l which blocks J_i inherits the current priority $p_i(t)$ of J_i.
  - J_l executes at its inherited priority $p_i(t)$ until the time t' where it releases every resource whose priority ceiling is equal to or higher than $p_i(t)$.
  - At that time, the priority of J_l returns to its priority $p_l(t')$ at the time t' when it was granted the resource(s).

Priority Ceiling Protocol



Comparison: Priority Inheritance vs Priority Ceiling Protocol

- Priority-Inheritance protocol is greedy: A requesting job will always get access to a resource, if this resource is free
- Priority-Ceiling protocol is non-greedy: A requesting job may not get access to a resource, though the resource is free (see additional allocation rule)

Priority Ceiling Protocol: Blocking Types

- Three ways in which a job J_h can be blocked by a low-priority job J_l
  - Direct Blocking: A job can be directly blocked by a lower-priority job that owns the requested resource
  - Priority Inheritance Blocking: A job J_l can be blocked by a lower-priority job J_l that has inherited the priority of a higher-priority job J_h
  - Avoidance Blocking: A job J is blocked by a lower-priority job J_l when J requests a resource S_x that is free at that time, and where J_l currently holds another resource S_y whose priority ceiling is equal to or higher than J's current priority $p(t)$.

Resource Access Protocols Priority Ceiling Protocol

Priority Ceiling Protocol: Important Properties

Freedom from Deadlock (Theorem 8.1 (Liu, 2000))

When resource accesses of a system of preemptive, priority-driven jobs on one processor are controlled by the priority-ceiling protocol, deadlock can never occur.

Priority Ceiling Protocol: Important Properties

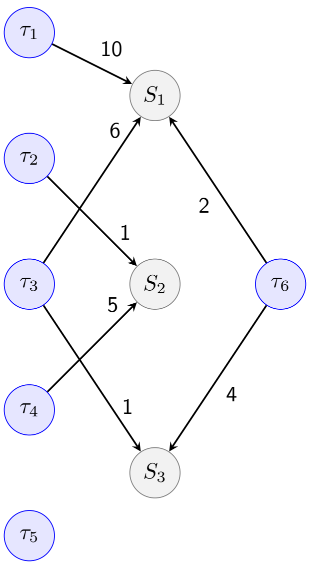
Freedom from Deadlock (Theorem 8.1 (Liu, 2000))

When resource accesses of a system of preemptive, priority-driven jobs on one processor are controlled by the priority-ceiling protocol, deadlock can never occur.

Blocking Duration (Theorem 8.2 (Liu, 2000))

When resource accesses of preemptive, priority-driven jobs on one processor are controlled by the priority-ceiling protocol, a job can be blocked for at most the duration of one critical section.

Blocking Time Calculation



	Direct Blocking by					Priority Inheritance Blocking by					Priority Ceiling Blocking by				
	J_2	J_3	J_4	J_5	J_6	J_2	J_3	J_4	J_5	J_6	J_2	J_3	J_4	J_5	J_6
J_1		6			2										
J_2	-		5			-	6			2	-	6			2
J_3		-			4		-	5		2		-	5		2
J_4			-					-		4			-		4
J_5				-					-	4				-	

The maximum blocking times are $B_1 = 6$, $B_2 = 6$, $B_3 = 5$, $B_4 = 4$, $B_5 = 4$.

Practical Factors: Blocking Time

An additional term for the blocking time B_i can be used in the time demand or response time analysis function. The term B_i reflects the blocking time, which a high priority job can experience when a low priority job executes a non-preemptable section, for instance a critical section protected by a semaphore.

- Time Demand Analysis: $w_i(t) = B_i + C_i + \sum_{k=1}^{i-1} \left\lceil \frac{t}{T_k} \right\rceil C_k, for 0 < t \leq T_i$
- Response Time Analysis: $R_i = B_i + C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j$

Resource Access Protocols Priority Ceiling Protocol

References

Giorgio C. Buttazzo. Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications. Springer, 3rd edition, 2011.

Jane W. S. Liu. Real-Time Systems. Prentice Hall, 2000.