Topic II: Resources and Resource Access Control

謝仁偉 教授
jenwei@mail.ntust.edu.tw
國立台灣科技大學 資訊工程系
2017 Fall

Outline

Assumptions

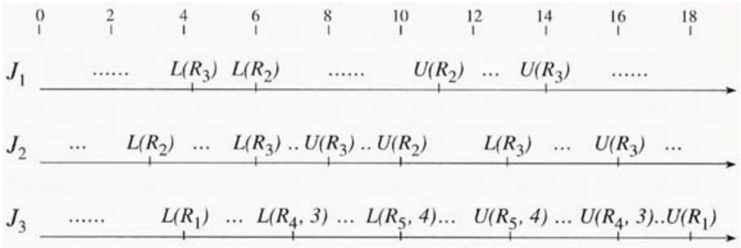
- The system contains only one processor.
- The system contains ρ types of serially reusable resources named $R_1, R_2, \ldots, R_\rho$.
  - There are v_i indistinguishable units of resource (of type) R_i, for $1 \le i \le \rho$.
  - Serially reusable resources are typically allocated to jobs on a nonpreemptive basis and used in a mutual exclusive manner.

Enforcement of Mutual Exclusion and Critical Sections (1/3)

- When a job wants to use η_i units of resource R_i, it executes a lock to request them, denoted by $L(R_i, \eta_i)$.
- When the job no longer needs the resource, it releases the resource by executing an unlock, denoted by $U(R_i, \eta_i)$.
- When a resource R_i has only one unit, we use the simpler notations $L(R_i)$ and $U(R_i)$ for lock and unlock.

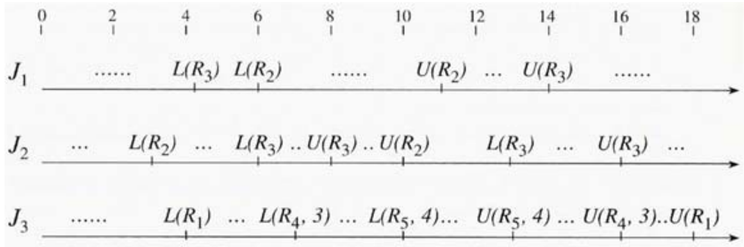
Enforcement of Mutual Exclusion and Critical Sections (2/3)

- We call a segment of a job that begins at a lock and ends at a matching unlock a critical section.
  - Since resources are released in the LIFO order, overlapping critical sections are properly nested.
  - A critical section that is not included in other critical sections is an outermost critical section.



Enforcement of Mutual Exclusion and Critical Sections (3/3)

- We denote each critical section by $[R, \eta; e]$.
  - R : the name of the resource
  - η : the number of units of the resource
  - e : the (maximum) execution time of the critical section
  - When $\eta = 1$, we use the simpler notation $[R; e]$.
  - We denote nested critical sections by nested square brackets.



Outline

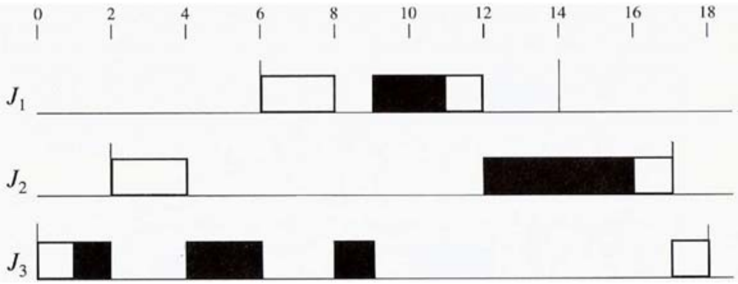
- Assumptions on Resources and Their Usage
- ***Effects of Resource Contention and Resource Access Control***
- Nonpreemptive Critical Sections Protocol
- Basic Priority-Inheritance Protocol
- Basic Priority-Ceiling Protocol
- Stack-Based, Priority-Ceiling (Ceiling-Priority) Protocol
- Preemption-Ceiling Protocols
- Controlling Accesses to Multiple-Unit Resources

Resource Access-Control Protocol

- Effects of resource contention
  - Priority inversion
  - Timing anomalies
  - Deadlock
- A resource access-control protocol is a set of rules that govern
  - When and under what conditions each request for resource is granted, and
  - How jobs requiring resources are scheduled

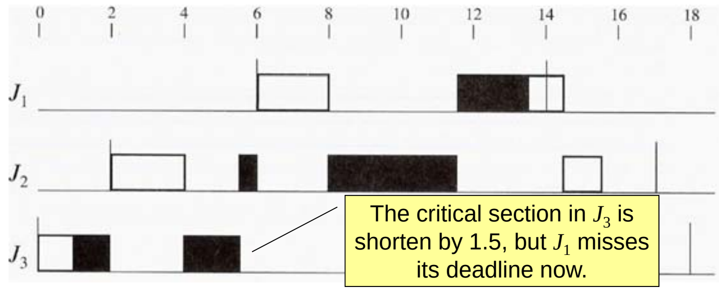
Priority Inversion

- Priority inversion can occur when the execution of some jobs or portions of jobs is nonpreemptable.
- Resource contentions among jobs can also cause priority inversion.



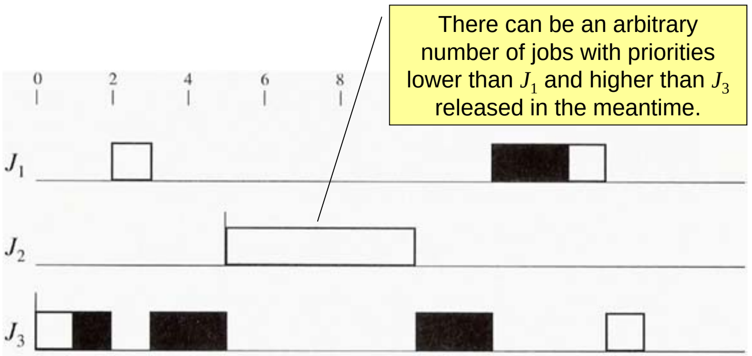
Timing Anomalies

- When priority inversion occurs, timing anomalies invariably follow.
- Timing anomaly: When we shorten the execution time of some critical section, rather than complete sooner, some job might complete later.



Unbounded Priority Inversion

- Without good resource access control, the duration of a priority inversion can be unbounded.



Deadlock

- Nonpreemptivity of resource allocation can also lead to deadlocks.
  - Two jobs both require resources X and Y. The jobs are in deadlock when one of them holds X and requests for Y, while the other holds Y and requests for X.

Criterion of Resource Access-Control Protocols

- No resource access-control protocol can eliminate the priority inversion and anomalous behavior caused by resource contention.
- A criterion we use to measure the performance of a resource access-control protocol is the blocking time of each job.
- ➡ A good resource access-control protocol should control priority inversions and prevent deadlock and keep the blocking time of every job bounded from the above.

Terms and Assumptions

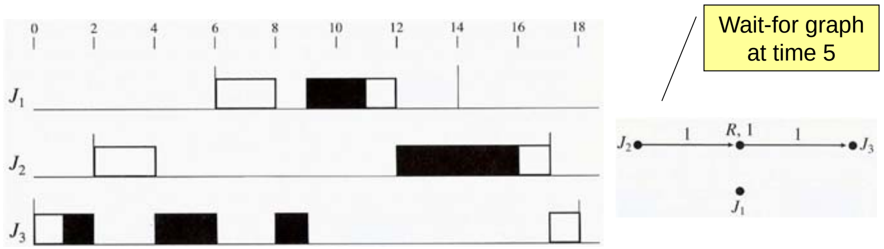
- **Assumption:** No job ever suspends itself and every job is preemptable on the processor.
- A higher-priority job J_h is said to be directly blocked by a lower-priority job J_l when J_l holds some resource which J_h requests and is not allocated.
- We sometimes denote the periodic task by the tuple $(\phi_i, p_i, e_i, D_i, [R, x; y])$.
  - In general, when jobs in the task require more than one resource and hence have more than one critical section, we put all the critical sections in the tuple.

Wait-for Graph (1/2)

- In the wait-for graph of a system
  - Every job that requires some resource: A vertex labeled by the name of the job.
  - Every resource: A vertex labeled by the name and the number of units of the resource.
  - Ownership edge: An edge with label x from a resource vertex to a job vertex if x units of the resource are allocated to the job at the time.
  - Wait-for edge: An edge with label y from a job vertex to a resource vertex if the job requested y units of the resource earlier and the request was denied.

Wait-for Graph (2/2)

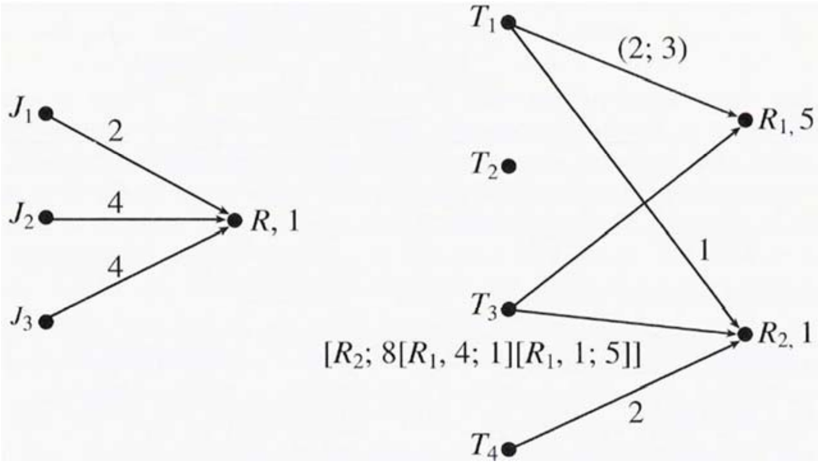
- A path in a wait-for graph from a higher-priority job to a lower-priority job represents the fact that the former is directly blocked by the later.
- A cyclic path in a wait-for graph indicates a deadlock.



Resource Requirements (1/2)

- We will specify resource requirements of a system by a bipartite graph
  - There is a vertex for every job and every resource named by the name of the job (or task) or resource it represents.
  - The integer next to each resource vertex R_i gives the number v_i of units of the resource.
  - A job J (or task T) requires a resource R_i is represented by an edge from the job (or task) vertex to the resource vertex.
  - We may label each edge by 2-tuple: (the number of units used in the critical section, the duration of the critical section)

Resource Requirements (2/2)



Outline

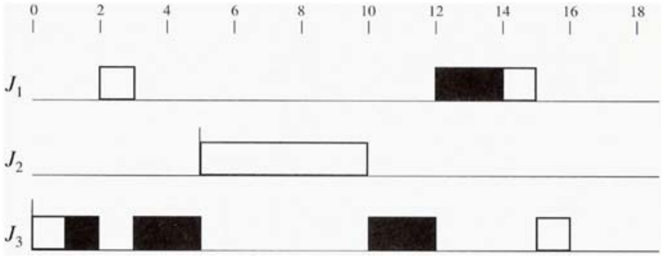
- Assumptions on Resources and Their Usage
- Effects of Resource Contention and Resource Access Control
- ***Nonpreemptive Critical Sections Protocol***
- Basic Priority-Inheritance Protocol
- Basic Priority-Ceiling Protocol
- Stack-Based, Priority-Ceiling (Ceiling-Priority) Protocol
- Preemption-Ceiling Protocols
- Controlling Accesses to Multiple-Unit Resources

Nonpreemptive Critical Sections

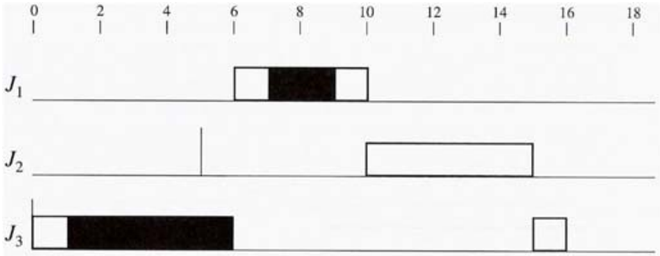
- Nonpreemptive Critical Section (NPCS) protocol
  - The simplest way to control access of resources
  - When a job requests a resource, it is always allocated the resource.
  - When a job holds any resource, it executes at a priority higher than the priority of all jobs.
  - No job is ever preempted when it hold any resource, deadlock can never occur.
  - J_h can be blocked only once, and its blocking time due to resource conflict is at most equal to the maximum execution time of the critical sections of all lower-priority jobs.

Example

Uncontrolled Priority Inversion



NPCS



Pros and Cons of NPCS

- The advantage is its simplicity, especially when the numbers of resource units are arbitrary.
  - Does not need any prior knowledge about resource requirements of jobs
  - Simple to implement and can be used in both fixed-priority and dynamic-priority systems
  - Good protocol when all the critical sections are short and when most of the jobs conflict with each other
- The shortcoming is that every job can be blocked by every lower-priority job that requires some resource even when there is no resource conflict between them.

Outline

- Assumptions on Resources and Their Usage
- Effects of Resource Contention and Resource Access Control
- Nonpreemptive Critical Sections Protocol
- ***Basic Priority-Inheritance Protocol***
- Basic Priority-Ceiling Protocol
- Stack-Based, Priority-Ceiling (Ceiling-Priority) Protocol
- Preemption-Ceiling Protocols
- Controlling Accesses to Multiple-Unit Resources

Basic Priority-Inheritance Protocol (PIP)

- A simple protocol proposed by Sha, *et al.**
- Work with any preemptive, priority-driven scheduling algorithm
- Does not require prior knowledge on resource requirements of jobs
- Does NOT prevent deadlock
- Assume every resource has only 1 unit

*Sha, L., R. Rajkumar, and J. P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Transactions on Computers*, Vol. 39, 1990.

Definition of Basic PIP

- We call the priority that is assigned to a job according to the scheduling algorithm its assigned priority.
- At any time t, each ready job J_l is scheduled and executes at its current priority $\pi_l(t)$, which may differ from its assigned priority and may vary with time.
- The current priority $\pi_l(t)$ of a job J_l may be raised to the higher priority $\pi_h(t)$ of another job J_h. We say that the lower-priority job J_l inherits the priority of the higher priority job J_h and that J_l executes at its inherited priority $\pi_h(t)$.

Rules of the Basic PIP (1/2)

1. **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 $\pi(t)$ of every job J is equal to its assigned priority. The job remains at this priority except under the condition stated in rule 3.
2. **Allocation Rule:** When a job J requires a resource R at time t,
   a) If R is free, R is allocated to J until J releases the resource
   b) If R is not free, the request is denied and J is blocked

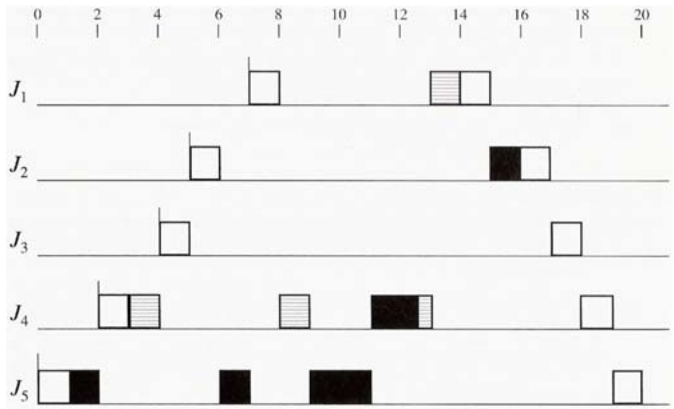
Rules of the Basic PIP (2/2)

3. **Priority-Inheritance Rule:** When the requesting job J becomes blocked, the job J_l which blocks J inherits the current priority $\pi(t)$ of J. The job J_l executes at its inherited priority $\pi(t)$ until it releases R; at that time, the priority of J_l returns to its priority $\pi_l(t')$ at the time t' when it acquires the resource R.

- The protocol ensures that the duration of priority inversion is never longer than the duration of an outermost critical section each time a job is blocked.

Example

Job	r_i	e_i	π_i	Critical Sections
J_1	7	3	1	[*Shaded*; 1]
J_2	5	3	2	[*Black*; 1]
J_3	4	2	3	
J_4	2	6	4	[*Shaded*; 4 [*Black*; 1.5]]
J_5	0	6	5	[*Black*; 4]



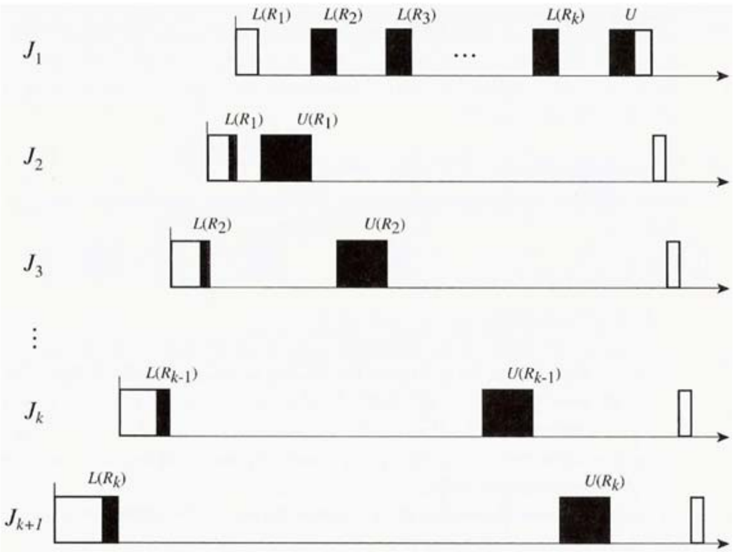
Properties of the PIP (1/2)

- There are two types of blocking:
  - Direct blocking: J_2 is directly blocked by J_5 in (6, 11] and by J_4 in (11, 12.5]
  - Priority-inheritance blocking (or simply inheritance blocking): J_3 is blocked by J_5 in (6, 7] because the latter inherits a higher priority in this interval.
  - Priority-inheritance blocking suffered by jobs that are not involved in resource contention is the cost for controlling the durations of priority inversion suffered by jobs that are involved in resource contention.

Properties of the PIP (2/2)

- Jobs can transitively block each other:
  - At time 9, J_5 blocks J_4, and J_4 blocks J_1.
  - In the time interval (9, 11), J_5 inherits J_4's priority, which J_4 inherited from J_1.
- Disadvantages of the PIP:
  - Does not prevent deadlock: suppose J_5 request the resource *Shaded* at time 6.5.
  - Does not reduce the blocking times suffered by jobs as small as possible.

Worst-Case Blocking Scenario for PIP



Outline

- Assumptions on Resources and Their Usage
- Effects of Resource Contention and Resource Access Control
- Nonpreemptive Critical Sections Protocol
- Basic Priority-Inheritance Protocol
- ***Basic Priority-Ceiling Protocol***
- Stack-Based, Priority-Ceiling (Ceiling-Priority) Protocol
- Preemption-Ceiling Protocols
- Controlling Accesses to Multiple-Unit Resources

Basic Priority-Ceiling Protocol (PCP)

- The priority-ceiling protocol* extends the priority-inheritance protocol to prevent deadlocks and to further reduce the blocking time.
- 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.

*Sha, L., R. Rajkumar, and J. P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Transactions on Computers*, Vol. 39, 1990.

Definitions

- The priority ceiling of any resource R_i is the highest priority of all the jobs that require R_i and is denoted by $\Pi(R_i)$.
- At any time t, the current priority ceiling (or the ceiling) $\hat{\Pi}(t)$ 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.
- If all the resource are free at the time, the current ceiling $\hat{\Pi}(t)$ is equal to Ω, a nonexisting priority level that is lower than the lowest priority of all jobs.

Rules of Basic Priority-Ceiling Protocol (1/3)

1. **Scheduling Rule:**
   a) At its release time t, the current priority $\pi(t)$ of every job J is equal to its assigned priority. The job remains at this priority except under the condition stated in rule 3.
   b) Every ready job J is scheduled preemptively and in a priority-driven manner at its current priority $\pi(t)$.

Rules of Basic Priority-Ceiling Protocol (2/3)

2. **Allocation Rule:** Whenever a job J requests a resource R at time t, one of the following two conditions occurs:
   a) The resource R is held by another job. J's request fails and J becomes blocked.
   b) The resource R is free.
      1) If J's priority $\pi(t)$ is higher than the current priority ceiling $\hat{\Pi}(t)$, R is allocated to J.
      2) If J's priority $\pi(t)$ is not higher than the ceiling $\hat{\Pi}(t)$ of the system, R is allocated to J only if J is the job holding the resource(s) whose priority ceiling is equal to $\hat{\Pi}(t)$; otherwise, J's request is denied, and J becomes blocked.

Rules of Basic Priority-Ceiling Protocol (3/3)

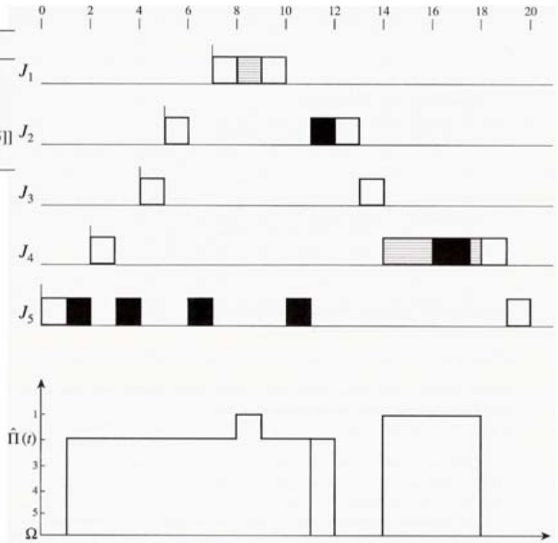
3. **Priority-Inheritance Rule:** When J becomes blocked, the job J_l which blocks J inherits the current priority $\pi(t)$ of J. J_l executes at its inherited priority until the time when it releases every resource whose priority ceiling is equal to or higher than $\pi(t)$; at that time, the priority of J_l returns to its priority $\pi_l(t')$ at the time t' when it was granted the resource(s).

Assumptions in Rules

- We note that 2) in part b) of rule 2 assumes that only one job holds all the resources with priority ceiling equal to $\hat{\Pi}(t)$.
- Rule 3 assumes that only one job is responsible for J's request being denied, because it holds either the requested resource or a resource with priority ceiling $\hat{\Pi}(t)$.

Example

Job	r_i	e_i	π_i	Critical Sections
J_1	7	3	1	[*Shaded*; 1]
J_2	5	3	2	[*Black*; 1]
J_3	4	2	3	
J_4	2	6	4	[*Shaded*; 4 [*Black*; 1.5]]
J_5	0	6	5	[*Black*; 4]

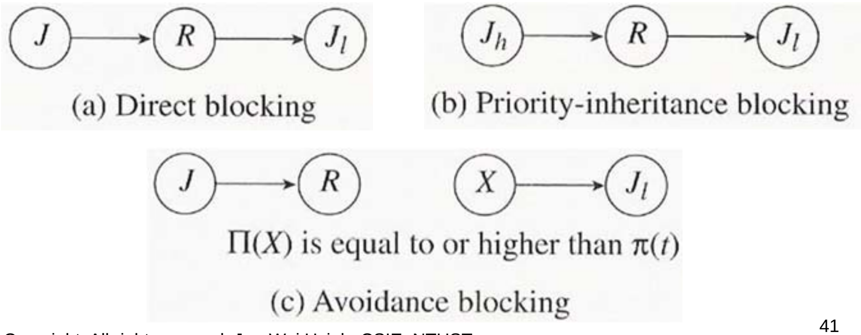


PIP vs. PCP

- A fundamental difference is that the PIP is greedy while the PCP is not.
- The priority-inheritance rules of these two protocols are essentially the same.
  - The difference arises because of the nongreedy nature of the PCP.
  - It is possible for J to be blocked by a lower-priority job which does not hold the requested resource according to the PCP, while this is impossible according to the PIP.

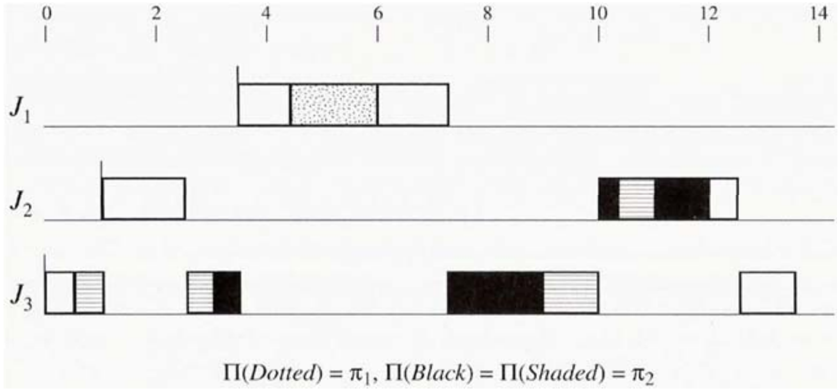
Ways for a Job to Block Another Job in PCP

- Priority-ceiling blocking (avoidance blocking): the requesting job J is blocked by a lower-priority job J_l when J requests a resource R that is free at the time. The reason is that J_l holds another resource X whose priority ceiling is equal to or higher than J's priority $\pi(t)$.



Deadlock Avoidance by PCP (1/3)

- The set of priority ceilings of resources impose a linear order on all the resources.



Deadlock Avoidance by PCP (2/3)

- At any time t, the priority $\pi(t)$ of a job J being higher than the current ceiling $\hat{\Pi}(t)$ of the system means that
  - Job J will not require any of the resources in use at t
  - Jobs with priorities equal to or higher than J will not require any of these resource
  - The priority ceiling $\hat{\Pi}(t)$ of the system tells us the subset of all jobs to which we can safely grant free resources at time t; this subset contains all the jobs that have higher priorities than $\hat{\Pi}(t)$.

Deadlock Avoidance by PCP (3/3)

- 2) in part b) of rule 2 states an exception to the rule that J's request for any resource is denied if its priority is not higher than the ceiling of the system.
  - The exception applies when J is the job holding the resource(s) whose priority ceiling(s) is equal to $\hat{\Pi}(t)$;
- **Theorem 1.** 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.

Duration of Blocking

- Under the PCP, a job may be directly blocked, avoidance blocked, and inheritance blocked by lower-priority jobs.
- Whether the PCP would cause a job to be blocked for a longer duration than the PIP (as a cost of its ability to prevent deadlock)?
- **Theorem 2.** When resource accesses of preemptive, priority-driven jobs on one processor are controlled by the PCP, a job can be blocked for at most the duration of one critical section.

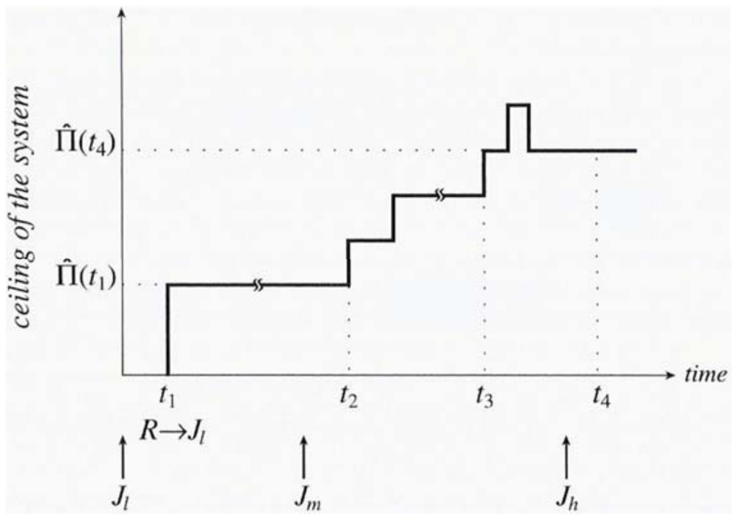
Informal Proof of Theorem 2

There are two parts to this argument:

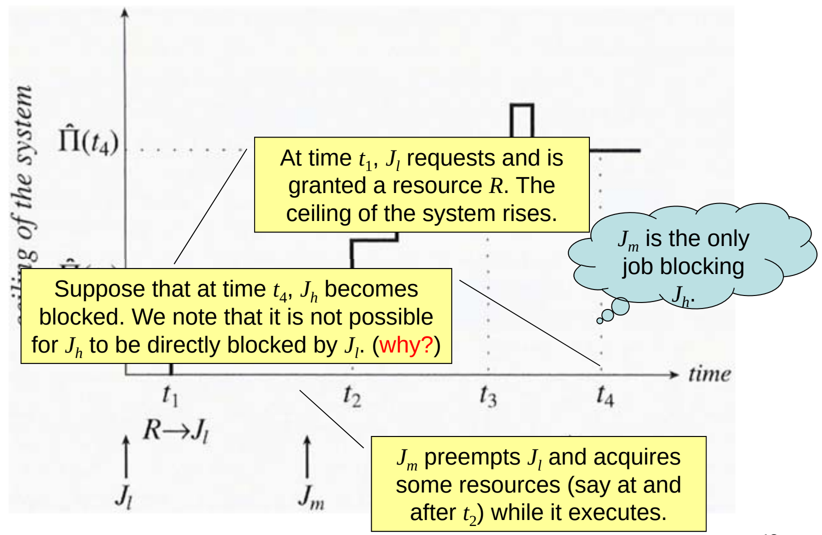
1. When a job becomes blocked, it is blocked by only one job
2. A job which blocks another job cannot be blocked in turn by some other job.

- There can be no transitive blocking under the priority-ceiling protocol.

Why 1. Is True?

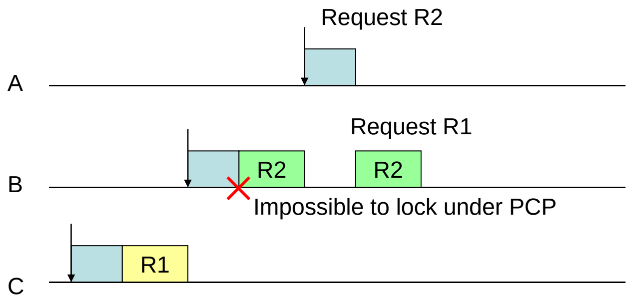


Why 1. Is True?



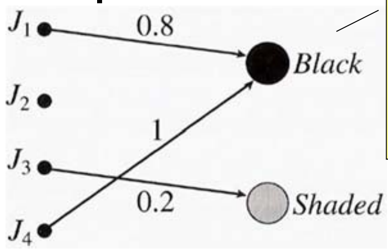
Why 2. Is True?

- Suppose three jobs J_l, J_m, and J_h are blocked transitively.



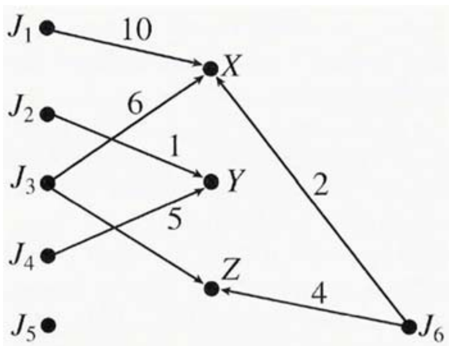
Computation of Blocking Time

- Theorem 2 makes it easy for us to compute an upper bound to the amount of time a job may be blocked due to resource conflicts.
  - We call this upper bound the blocking time (due to resource conflicts) of the job.
- **Example:**

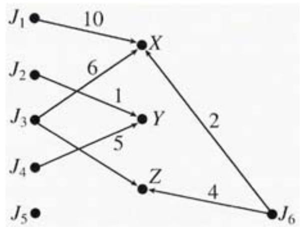


Although J_2 and J_3 do not require the resource *Black*, they can be priority-inheritance blocked by J_4 since J_4 can inherit priority π_1. Hence, the blocking time $b_2(rc)$ and $b_3(rc)$ are also one.

A Slightly More Complicated Example

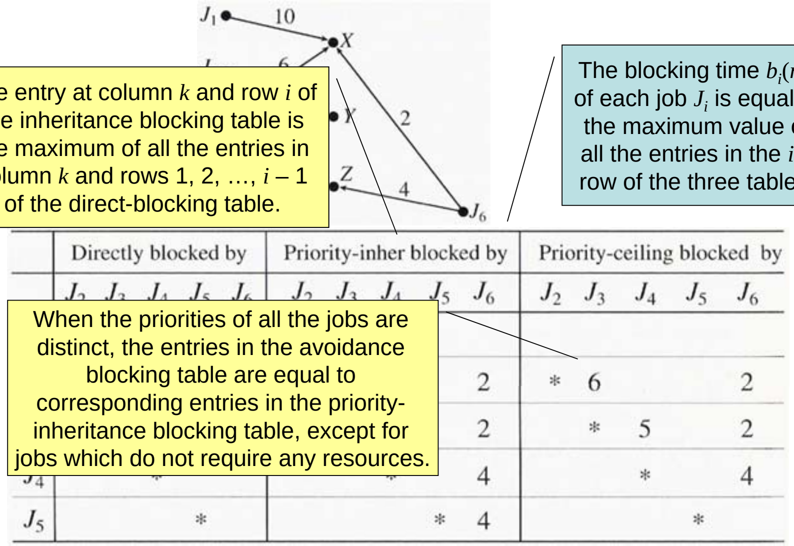


A Systematic Way to Compute Blocking



	Directly blocked by					Priority-inher blocked by					Priority-ceiling blocked 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				*	

A Systematic Way to Compute Blocking



The entry at column k and row i of the inheritance blocking table is the maximum of all the entries in column k and rows 1, 2, …, $i-1$ of the direct-blocking table.

The blocking time $b_i(rc)$ of each job J_i is equal to the maximum value of all the entries in the ith row of the three tables.

When the priorities of all the jobs are distinct, the entries in the avoidance blocking table are equal to corresponding entries in the priority-inheritance blocking table, except for jobs which do not require any resources.

Outline

- Assumptions on Resources and Their Usage
- Effects of Resource Contention and Resource Access Control
- Nonpreemptive Critical Sections Protocol
- Basic Priority-Inheritance Protocol
- Basic Priority-Ceiling Protocol
- ***Stack-Based, Priority-Ceiling (Ceiling-Priority) Protocol***
- Preemption-Ceiling Protocols
- Controlling Accesses to Multiple-Unit Resources

Stack-Based, Priority-Ceiling (Ceiling-Priority) Protocol

- We will give two different definitions of a protocol
  - Simpler than the PCP
  - Has the same worst-case performance as the PCP
- The different definitions arise from two different motivations:
  - To provide stack-sharing capability (Stack-Based, Priority-Ceiling Protocol)
  - To simplify the priority-ceiling protocol (Ceiling-Priority Protocol)

Motivation of Stack-Sharing Priority-Ceiling Protocol

- Sometimes, especially in systems where the number of jobs is large, it may be necessary for the jobs to share a common run-time stack, in order to reduce overall memory demand.

How Stack Sharing Operate

- Space in the (shared) stack is allocated to jobs contiguously in the last-in-first-out manner.
- When a job J executes, its stack space is on the top of the stack.
- The space is freed when the job completes.
- When J is preempted, the preempting job has the stack space above J's.
- J can resume execution only after all the jobs holding stack space above its space complete, free their stack space, and leave J's stack space on the top of the stack again.

Rules Defining Basic Stack-Based, Priority-Ceiling Protocol

1. **Update of the Current Ceiling:** Whenever all the resources are free, the ceiling of the system is Ω. The ceiling $\hat{\Pi}(t)$ is updated each time a resource is allocated or freed.
2. **Scheduling Rule:** After a job is released, it is blocked from starting execution until its assigned priority is higher than the current ceiling $\hat{\Pi}(t)$ of the system.
3. **Allocation Rule:** Whenever a job requests a resource, it is allocated the resource.

Stack-Based, Priority-Ceiling Protocol

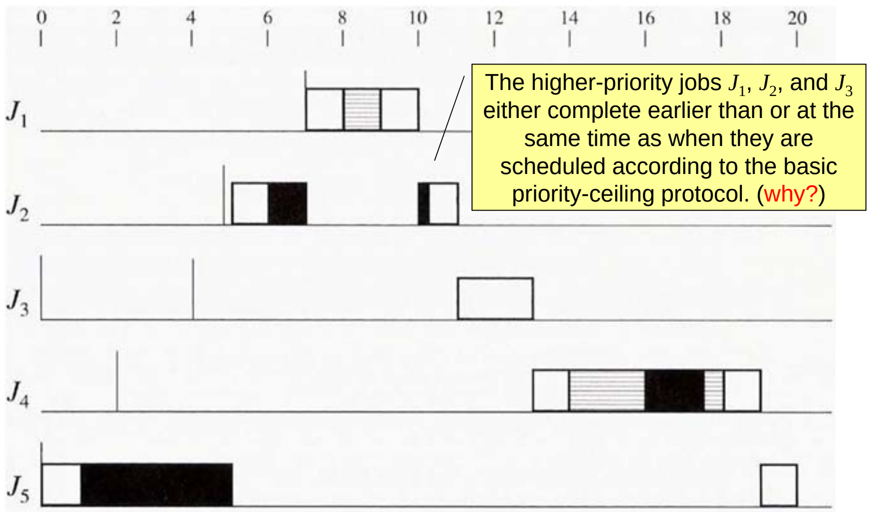
- According to the scheduling rule, when a job begins to execute, all the resource it will ever need during its execution are free. (why?)

➡ No job is ever blocked once its execution begins.

- When a job J is preempted, all the resources the preempting job will require are free, ensuring that the preempting job can always complete so J can resume.

➡ Deadlock can never occur.

Example (vs. PCP)



Rules Defining the Ceiling-Priority Protocol

1. **Scheduling Rule:**
   a) Every job executes at its assigned priority when it does not hold any resource. Jobs of the same priority are scheduled on the FIFO basis.
   b) The priority of each job holding any resource is equal to the highest of the priority ceiling of all resources held by the job.
2. **Allocation Rule:** Whenever a job requests a resource, it is allocated the resource.

- Note that when jobs never self-suspend, the stack-based priority-ceiling and ceiling-priority protocols are the same.

Blocking Time and Context-Switch Overhead

- **Theorem 3.** The longest blocking time suffered by every job is the same for the stack-based and basic priority-ceiling protocols.
- The context-switch overhead is smaller under the stack-based version.
  – Because no job is ever blocked once its execution starts, no job ever suffers more than two context switches.

Outline

- Assumptions on Resources and Their Usage
- Effects of Resource Contention and Resource Access Control
- Nonpreemptive Critical Sections Protocol
- Basic Priority-Inheritance Protocol
- Basic Priority-Ceiling Protocol
- Stack-Based, Priority-Ceiling (Ceiling-Priority) Protocol
- ***Preemption-Ceiling Protocols***
- Controlling Accesses to Multiple-Unit Resources

Motivation

- In a dynamic-priority system, the priorities of the periodic tasks change with time while the resources required by each task remain constant.
- The priority ceilings of the resources may change with time.
- The preemption-ceiling protocol is based on the clever observation that the potentials of resource contentions in a dynamic-priority system do not change with time, just as in fixed-priority systems, and hence can be analyzed statically.

Observations (1/2)

- The fact that a job J_h has a higher priority than another job J_l and they both require some resource does not imply that J_l can directly block J_h. This blocking can occur only when it is possible for J_h to preempt J_l.
- When determining whether a free resource can be granted to a job, it is not necessary to be concerned with the resource requirements of all higher-priority jobs; only those that can preempt the job.

Observations (2/2)

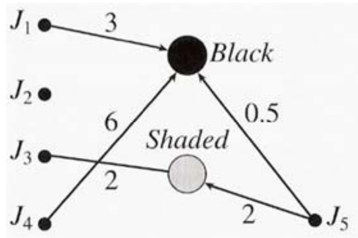
- For some dynamic priority assignments, it is possible to determine a priori the possibility that jobs in each periodic task will preempt the jobs in other periodic tasks.
- In a deadline-driven system, no job in a periodic task with a smaller relative deadline is ever preempted by jobs in periodic tasks with identical or larger relative deadlines.

Preemption Levels (1/4)

- The possibility that a job J_i will preempt another job is captured by the parameter preemption level ψ_i of J_i.
  - The preemption levels of jobs are functions of their priorities and release times.
  - According to a valid preemption-level assignment, for every pair of jobs J_i and J_k, the preemption level ψ_i of J_i being equal to or higher than the preemption level ψ_k of J_k implies that it is never possible for J_k to preempt J_i.
- **Validity Condition:** If π_i is higher than π_k and $r_i > r_k$, then ψ_i is higher than ψ_k.

Preemption Levels (2/4)

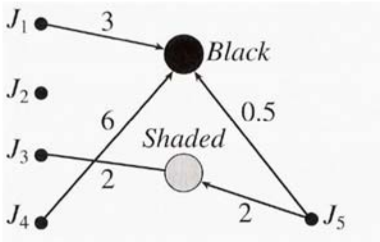
- The preemption levels of jobs that are not given by the above rule are valid as long as the linear order over all jobs defined by preemption-level assignment does not violate the validity condition.
- **Example:**



Jobs	r_i	e_i	π_i	ψ_i
J_1	5	5	1	1
J_2	7	3	2	1
J_3	4	4	3	2
J_4	0	9	4	3
J_5	3	4	5	3

Preemption Levels (3/4)

- **Example:**



Jobs	r_i	e_i	π_i	ψ_i
J_1	5	5	1	1
J_2	7	3	2	1
J_3	4	4	3	2
J_4	0	9	4	3
J_5	3	4	5	3

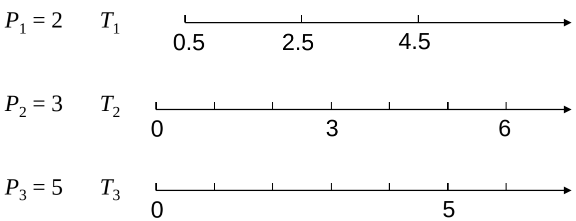
- Method 1: release time + priority (1, 1, 2, 3, 3)
- Method 2: release time (2, 1, 3, 5, 4)
- Method 3: relative deadline (EDF, periodic task)

Preemption Levels (4/4)

- A system of periodic tasks is a fixed preemption-level system if there is a valid assignment of preemption levels to all jobs such that the jobs in every task have the same preemption level.
  - All fixed-priority systems are also fixed preemption-level systems.
  - Periodic tasks scheduled on the LIFO basis have varying preemption levels. (illustrate)

Example: Varying Preemption Levels

- Periodic tasks scheduled on the LIFO basis



- How about preemption levels?
  - (0, 3):
  - (3, 4.5):
  - (4.5, 5):
  - (5, 6):

Definitions of Protocols

- A preemption-ceiling protocol makes decisions on whether to grant a free resource to any job based on the preemption level of the job in a manner similar to the priority-ceiling protocol.
  - Assumptions:
    - The resource requirements of all jobs are known a priori.
    - There is only 1 unit of each resource
  - The preemption ceiling $\psi(R)$ of a resource R is the highest preemption level of all the jobs that require the resource.
  - The (preemption) ceiling of the system $\hat{\Psi}(t)$ at any time t is the highest preemption ceiling of all the resources that are in use at t.

Rules of Preemption-Ceiling Protocol (1/2)

- Basic priority-ceiling and preemption-ceiling protocols differ mainly in their allocation rules.
- **Rules of Basic Preemption-Ceiling Protocol:**

1 and 3. The **Scheduling Rule** and **Priority Inheritance Rule** are the same as the corresponding rules of the PCP.

2. Allocation Rule: Whenever a job J requests resource R at time t, one of the following two conditions occurs:

Rules of Preemption-Ceiling Protocol (2/2)

a) The resource R is held by another job. J's request fails, and J becomes blocked.

b) The resource R is free.

1) If J's preemption level $\psi(t)$ is higher than the current preemption ceiling $\hat{\Psi}(t)$ of the system, R is allocated to J.

2) If J's preemption level $\psi(t)$ is not higher than the ceiling $\hat{\Psi}(t)$ of the system, R is allocated to J only if J is the job holding the resource(s) whose preemption ceiling is equal to $\hat{\Psi}(t)$; otherwise, J's request is denied, and J becomes blocked.

back

Stack-Based Protocol (SBP)

- The stack-based preemption-ceiling protocol is called Stack-Based Protocol (SBP) by Backer*.
- The difference is that priority levels/ceilings are replaced by preemption levels/ceilings.
  - Rules of Stack-Based Priority-Ceiling Protocol
- In addition, the stack-based preemption-ceiling protocol has an inheritance rule.

*Baker, T.P., "A Stack-Based Resource Allocation Policy for Real-Time Processes," *IEEE Real-Time System Symposium*, 1991.

Rules of Stack-Based Protocol (1/2)

0. Update of the Current Ceiling: Whenever all the resources are free, the preemption ceiling of the system is Ω. The preemption ceiling $\hat{\Psi}(t)$ is updated each time a resource is allocated or freed.

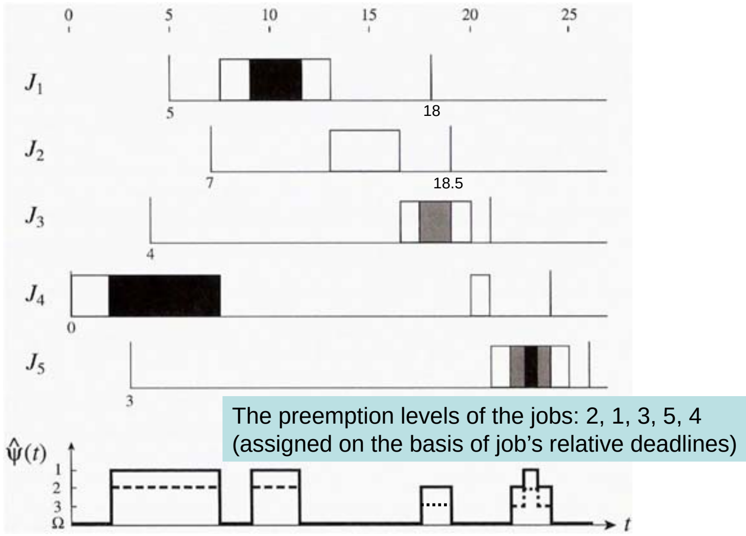
1. Scheduling Rule: After a job is released, it is blocked from starting execution until its preemption level is higher than the current ceiling $\hat{\Psi}(t)$ of the system and the preemption level of the executing job. At any time t, jobs that are not blocked are scheduled on the

Rules of Stack-Based Protocol (2/2)

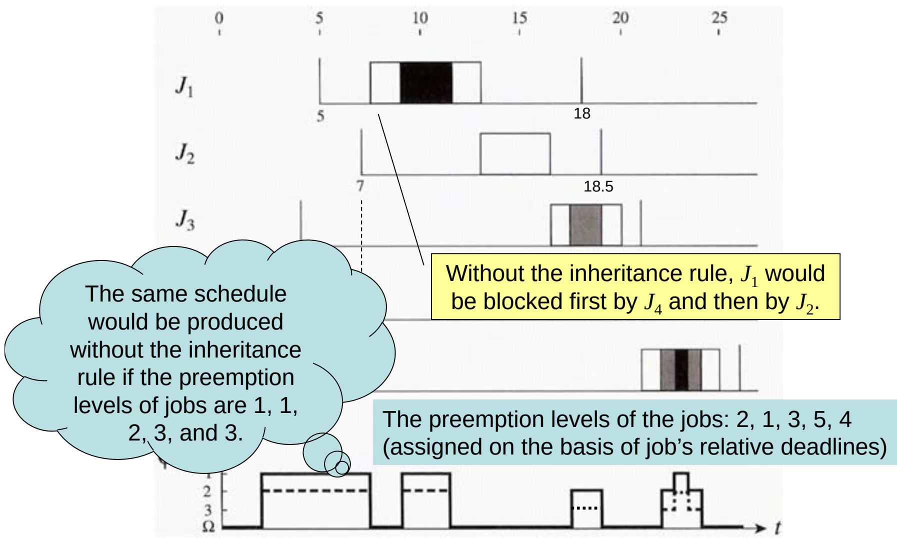
processor in a priority-driven, preemptive manner according to their assigned priorities.

2. **Allocation Rule:** Whenever a job J requests for a resource R, it is allocated resource.
3. **Priority-Inheritance Rule:** When some job is blocked from starting, the blocking job inherits the highest priority of all the blocked jobs.

Why We Need Priority-Inheritance Rule?



Why We Need Priority-Inheritance Rule?



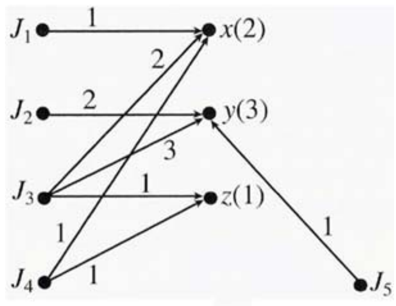
Outline

- Assumptions on Resources and Their Usage
- Effects of Resource Contention and Resource Access Control
- Nonpreemptive Critical Sections Protocol
- Basic Priority-Inheritance Protocol
- Basic Priority-Ceiling Protocol
- Stack-Based, Priority-Ceiling (Ceiling-Priority) Protocol
- Preemption-Ceiling Protocols
- ***Controlling Accesses to Multiple-Unit Resources***

Notations (1/2)

- We let $\Pi(R_i, k)$, for $k \le v_i$, denote the priority ceiling of a resource R_i when k out of the v_i ($\ge$ 1) units of R_i are free.
  - If one or more jobs in the system require more than k units of R_i, $\Pi(R_i, k)$ is the highest priority of all these jobs.
  - If no job requires more than k units of R_i, $\Pi(R_i, k)$ is equal to Ω, the nonexisting lowest priority.
  - The priority ceiling of a resource R_j that has only 1 unit is $\Pi(R_j, 0)$.
- Let $k_i(t)$ denote the number of units of R_i that are free at time t.

Example



	Resources	$x(2)$	$y(3)$	$z(1)$
Units Required by	J_1	1	0	0
	J_2	0	2	0
	J_3	2	3	1
	J_4	1	0	1
	J_5	0	1	0
	$\Pi(*, 0)$			
	$\Pi(*, 1)$			
	$\Pi(*, 2)$			
	$\Pi(*, 3)$			

Notations (2/2)

- The preemption ceiling $\psi(R_i, k)$ of the resource R_i when k units of R_i are free is the highest preemption level of all the jobs that require more than k units of R_i.
- The preemption ceiling of the system at time t is equal to the highest preemption ceiling of all the resources at the time.

Multiple-Unit Ceiling-Priority Protocol

- In essence, the scheduling and allocation rules remains unchanged except for the new definition of priority ceiling of resources.
- **Scheduling Rule 1b)** needs to be rephrased: Upon acquiring a resource R and leaving $k \ge 0$ free units of R, a job executes at the higher of its priority and the priority ceiling $\Pi(R, k)$ of R.

Multiple-Unit Priority-(Preemption-) Ceiling Protocol (1/2)

- Similarly, the allocation rule of the priority-ceiling (or preemption-ceiling) protocol for multiple units of resources is a straightforward modification.

Whenever a job J requests k units of resource R at time t, one of the following two condition occurs:

a) Less than k units of R are free. J's request fails and J becomes directly blocked.

b) k or more units of R are free.

   1) If J's priority $\pi(t)$ [preemption level $\psi(t)$] is higher than the current priority ceiling $\hat{\Pi}(t)$ [preemption ceiling $\hat{\Psi}(t)$] of the system at the time, k units of R are allocated to J until it releases them.

Multiple-Unit Priority-(Preemption-) Ceiling Protocol (2/2)

2) If J's priority $\pi(t)$ [preemption level $\psi(t)$] is not higher than the system ceiling $\hat{\Pi}(t)$ [$\hat{\Psi}(t)$], k units of R are allocated to J only if J holds the resource(s) whose priority ceiling (preemption ceiling) is equal to $\hat{\Pi}(t)$ [$\hat{\Psi}(t)$]; otherwise, J's request is denied, and J becomes blocked.

You can see that this rule is essentially the same as the allocation rule of the basic version. The only change is in the wording to accommodate multiple-unit requests.

Priority-Inheritance Rule (1/2)

- In the case where there is only 1 unit of each resource, we have shown that when a job J is blocked, only one lower-priority job is responsible for this blocking and this lower-priority job inherits J's priority.

➡ In a system containing multiple resource units, which job shall inherits the priority?

- **Example:** 3 units of resource R, 4 jobs each requiring 1 unit of R. J_1 request a unit of R, all 3 units are held by the other 3 jobs.

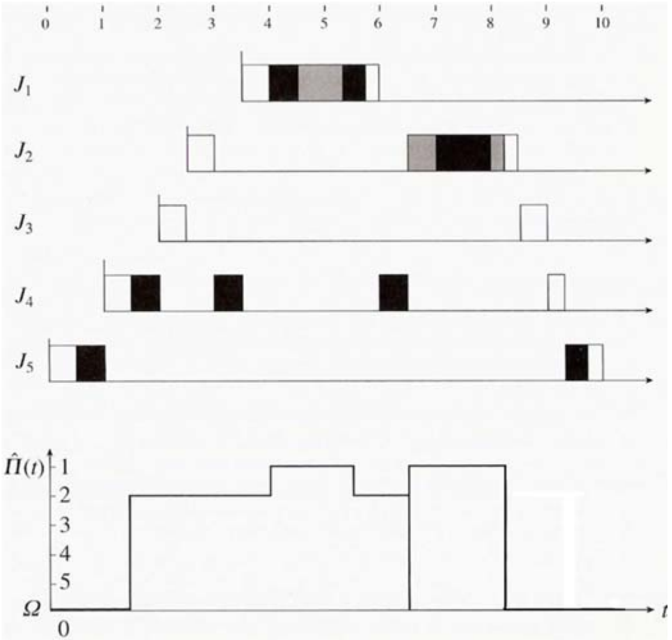
Priority-Inheritance Rule (2/2)

- **Priority-Inheritance Rule:** When the requesting job J becomes blocked at t, the job with the highest priority among all the jobs holding the resource R that has the highest priority ceiling among all the resources inherits J's priority until it releases its unit of R.

➡ With this rule, each job is blocked at most once for the duration of one critical section.

- **Example:**

	no. of units	units required					$\Pi(*, k)$, k =					
		J_1	J_2	J_3	J_4	J_5	0	1	2	3	4	5
Black	5	2	4	0	1	1	1	1	2	2	Ω	Ω
Shaded	1	1	1	0	0	1	1	Ω	Ω	Ω	Ω	Ω


0
1
2
3
4
5
6
7
8
9
10
J_1
J_2
J_3
J_4
J_5
$\hat{\Pi}(t)$
1
2
3
4
5
Ω
0
t