Resource Access Protocols

Prof. Dr. Jian-Jia Chen

LS 12, TU Dortmund

10 May 2016

Outline

Resource Access Protocols

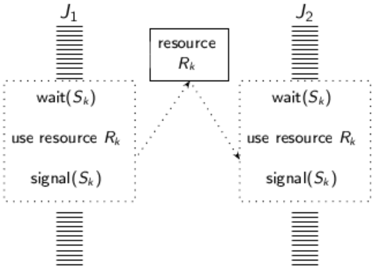
Priority Inheritance Protocol

Priority Ceiling Protocol

Why do We Have to Worry about Resource Sharing?

Shared Resources:

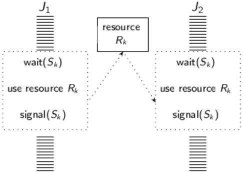
- Data structures, variables, main memory area, files, set of registers, I/O units, the processor, etc.
- Mutual exclusion, critical section
  - When a job enters the critical section of a shared resource, the accesses to the shared resource from other jobs are *blocked*.

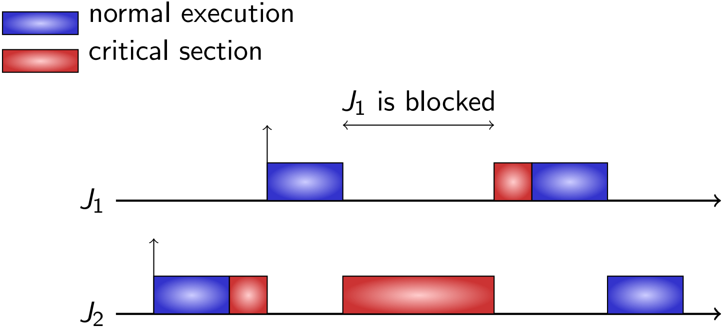


Priority Inversion

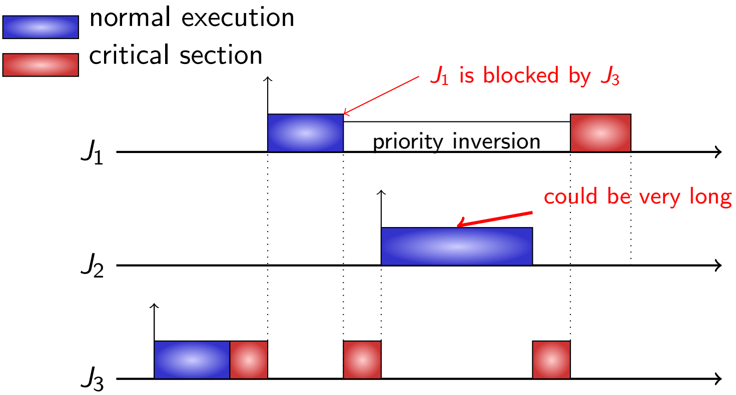
Priority Inversion: A higher priority job is *blocked* by a lower-priority job.

- Unavoidable when there are critical sections





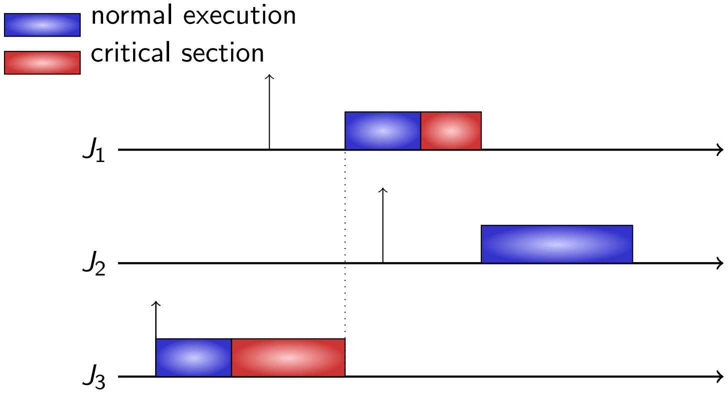
Priority Inversion: Another Example



Naïve Solution for Priority Inversion

Disallow preemption during critical sections

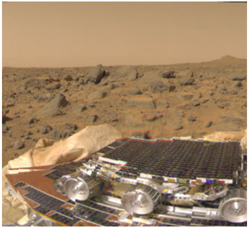
- It is simple
- But, it creates unnecessary blocking, as unrelated tasks may be blocked



Case Study: MARS Pathfinder Problem (1)

A few days into the mission.....

Not long after Pathfinder started gathering meteorological data, the spacecraft began experiencing total system resets, each resulting in losses of data.



Case Study: MARS Pathfinder Problem (2)

"VxWorks provides preemptive priority scheduling of threads. Tasks on the Pathfinder spacecraft were executed as threads with priorities that were assigned in the usual manner reflecting the relative urgency of these tasks."

"Pathfinder contained an information bus, which you can think of as a shared memory area used for passing information between different components of the spacecraft."

A bus management task ran frequently with high priority to move certain kinds of data in and out of the information bus. Access to the bus was synchronized with mutual exclusion locks (mutexes).

- The meteorological data gathering task ran as an infrequent, low priority thread, ... When publishing its data, it would acquire a mutex, do writes to the bus, and release the mutex.
- It also had a communications task that ran with medium priority.

Case Study: MARS Pathfinder Problem (3)

high priority	medium priority	low priority
data retrieval from memory	communication task	data collection

"Most of the time this combination worked fine. However, very infrequently it was possible for an interrupt to occur that caused the (medium priority) communications task to be scheduled during the short interval while the (high priority) information bus thread was blocked waiting for the (low priority) meteorological data thread. In this case, the long-running communications task, having higher priority than the meteorological task, would prevent it from running, consequently preventing the blocked information bus task from running. After some time had passed, a watchdog timer would go off, notice that the data bus task had not been executed for some time, conclude that something had gone drastically wrong, and initiate a total system reset. This scenario is a classic case of priority inversion."

Resource Access Protocols

- Spirit
  - Modify (increase) the priority of those tasks/jobs that cause blocking.
  - When a job J_j blocks one or more higher-priority jobs, it temporarily is assumed to have a higher priority.
- Methods
  - Priority Inheritance Protocol (PIP), for fixed-priority scheduling
  - Priority Ceiling Protocol (PCP), for fixed-priority scheduling
  - Stack Resource Policy (SRP), for both fixed- and dynamic-priority scheduling
  - others.....

Outline

Resource Access Protocols

Priority Inheritance Protocol

Priority Ceiling Protocol

Priority Inheritance Protocol (PIP)

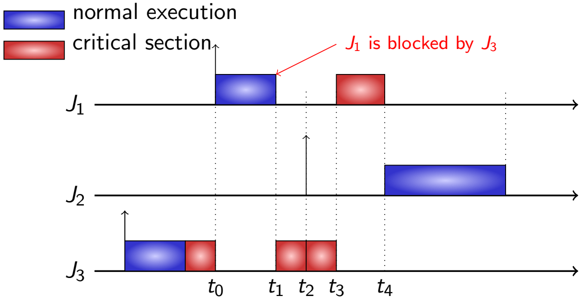
When a lower-priority job J_j blocks a higher-priority job, the priority of job J_j is *promoted* to the priority level of highest-priority job that job J_j blocks.

For example, if the priority order is $J_1 > J_2 > J_3 > J_4 > J_5$,

- When job J_4 blocks jobs J_2 and J_3, the priority of J_4 is promoted to the priority level of J_2.
- When job J_5 blocks jobs J_1 and J_3, the priority of J_5 is promoted to the priority level of J_1.

Priority inheritance solved the Mars Pathfinder problem: the VxWorks operating system used in the pathfinder implements a flag for the calls to mutex primitives. This flag allows priority inheritance to be set to on. When the software was shipped, it was set to off.

Example of PIP



- t_0: J_1 arrives and preempts J_3, since J_1 does not want to enter the critical section
- t_1: J_1 locks the semaphore and tries to enter the critical section. J_1 is blocked by J_3, and J_3 inherits J_1's priority
- t_2: J_2 arrives and has a lower priority than J_3, since J_3 inherited J_1's priority.
- t_3: J_3 leaves its critical section, and J_1 now preempts J_3.
- t_4: J_1 finishes, and J_2 is the highest-priority task.

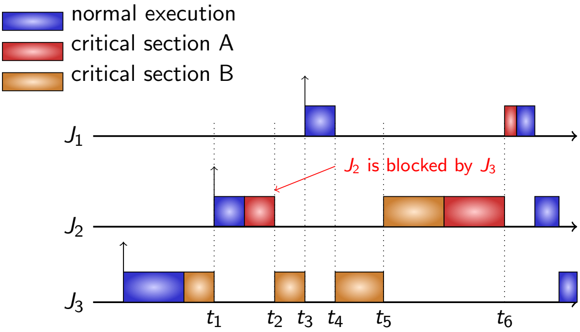
Blocking Properties of PIP (under Properly Nested)

Blocking in PIP

- *Direct Blocking*: higher-priority job tries to acquire a resource held by a lower-priority job.
- *Push-through Blocking*: medium-priority job is blocked by a lower-priority job that has a higher priority from a job it directly blocks
- *Transitive Blocking*: higher-priority job is blocked by a medium-priority job due to nested critical sections.

- Under PIP, if there are n lower priority jobs, a higher-priority job J_i can be blocked *as long as the duration of n* critical sections.
- Under PIP, if there are m distinct semaphores that can block a job J_i, then J_i can be blocked *as long as the duration of m* critical sections.
- Under PIP, a higher-priority job J_i can be blocked for at most $\min\{n, m\}$ critical sections.

Transitive Blocking



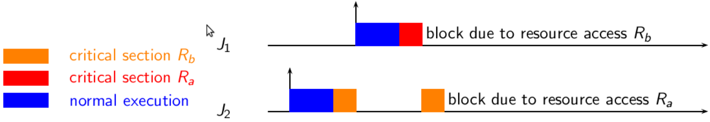
Three jobs with two semaphores A and B.

- J_1: normal execution, wait A, signal A, normal execution
- J_2: normal execution, wait A, wait B, signal B, signal A, normal execution
- J_3: normal execution, wait B, signal B, normal execution

Problem of PIP

Problems of PIP

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



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

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

Schedulability Test under Blocking Time

Definition

The Blocking Time B_i of a sporadic task τ_i is the maximum blocking time, due to *lower-priority jobs*, that a job of task τ_i may experience.

The time-demand function $W_i(t)$ of the task τ_i is redefined as follows:

$$W_i(t) = B_i + C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j.$$

Theorem

A system $\mathcal{T}$ of periodic, preemptable tasks with constrained deadlines is schedulable on one processor by a fixed-priority scheduling algorithm if

$$\forall \tau_i \in \mathcal{T},\ \exists t \text{ with } 0 < t \leq D_i \ \ \textit{and} \ \ W_i(t) \leq t$$

holds.

Schedulability Test under Blocking Time

Theorem

A system $\mathcal{T}$ of periodic, preemptable tasks is schedulable on one processor by RM if

$$\forall \tau_i \in \mathcal{T}, \left(U_i + \frac{B_i}{T_i} + 1\right) \cdot \prod_{j=1}^{i-1}(U_j + 1) \leq 2.$$

Theorem

A system $\mathcal{T}$ of periodic, preemptable tasks is schedulable on one processor by RM if

$$\forall \tau_i \in \mathcal{T} \; \frac{B_i}{T_i} + \sum_{j=1}^{i} U_j \leq i(2^{\frac{1}{i}} - 1).$$

Calculating the Blocking Time for PIP

In general, a bit complicated. We only focus on a special case, in which there are no nested critical sections (to avoid transitive blocking).

So, what we have to do is:

- Calculate the maximum duration of the blocking time due to lower-priority tasks
- Calculate the maximum duration of the blocking time due to guarded semaphores
- Use the minimum between these two values as the blocking time

Computing Blocking Time: Example

	$S_1()$	$S_2()$	$S_3()$
τ_1	1	2	0
τ_2	0	9	3
τ_3	8	7	0
τ_4	6	5	4

$S_j(\tau_i)$ is the worst-case execution time of a critical section guarded by semaphore "S_j" in task τ_i

	blocking time (by the lower-priority tasks)	blocking time (by the semaphores)	blocking time upper bound
τ_1	$9+8+6=23$	$8+9+4=21$	21
τ_2	$8+6=14$	$8+7+4=19$	14
τ_3	6	$6+5+4=15$	6
τ_4	0	0	0

Safe but not tight!!

Computing Blocking Time: Example

	$S_1()$	$S_2()$	$S_3()$
τ_1	1	2	0
τ_2	0	9	3
τ_3	8	7	0
τ_4	6	5	4

$S_j(\tau_i)$ is the worst-case execution time of a critical section guarded by semaphore "S_j" in task τ_i

A semaphore can block a task τ_i only when it is used by τ_i or other tasks with higher priority than τ_i

	blocking time (by the lower-priority tasks)	blocking time (by the semaphores)	blocking time upper bound
τ_1	$9+8+6=23$	$8+9=17$	17
τ_2	$8+6=14$	$8+7+4=19$	14
τ_3	6	$6+5+4=15$	6
τ_4	0	0	0

Safe and tighter, but still not tight!!

Outline

Resource Access Protocols

Priority Inheritance Protocol

Priority Ceiling Protocol

Priority Ceiling Protocol (PCP)

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

PCP

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

Properties of PCP

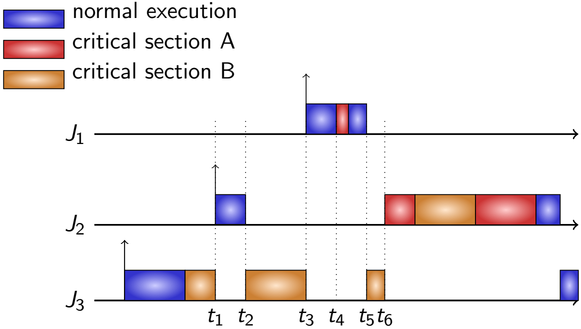
Theorem

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

Theorem

When the resource accesses of a system 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.

Example of PCP



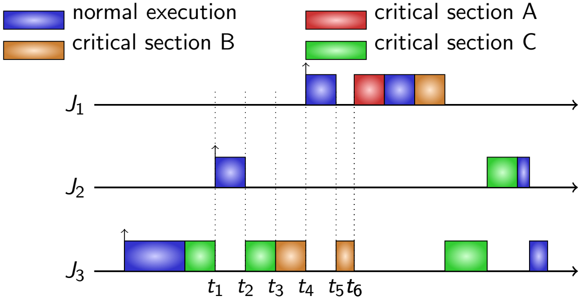
priority ceilings:

- A: priority level 1
- B: priority level 2

Three jobs with two semaphores A and B.

- J_1: normal execution, wait A, signal A, normal execution
- J_2: normal execution, wait A, wait B, signal B, signal A, normal execution
- J_3: normal execution, wait B, signal B, normal execution

Example of PCP (2nd)



priority ceilings:

- A: priority level 1
- B: priority level 1
- C: priority level 2

Three jobs with three semaphores A, B, and C.

- J_1: normal execution, wait A, signal A, normal execution, wait B, signal B, normal execution
- J_2: normal execution, wait C, signal C, normal execution
- J_3: normal execution, wait C, wait B, signal B, signal C, normal execution

Properties of PCP (cont.)

Theorem

The worst-case blocking time B_i for task τ_i is at most

$$\max_{j>i,R}\{C_{j,R} | \Pi(R) \leq i\},$$

where $C_{j,R}$ is the worst-case (consecutively) execution time resource R is required to execute an instance of task τ_j.

Quiz

Is the rate-monotonic scheduling algorithm still optimal for static-priority assignment when PCP is applied?