Resources and Resource Access Control

- Basic Priority-inheritance Protocol
  - Works with any preemptive, priority-driven scheduling algorithm
  - Does not require any prior knowledge of the jobs' resource requirements
  - Does NOT prevent deadlock
  - If one uses some other mechanism to prevent deadlock, it ensures that no job is ever blocked indefinitely due to uncontrolled priority inversion

Resources and Resource Access Control

- Assumptions (for all of the following protocols):
  - Each resource has only 1 unit
  - The priority assigned to a job according to the scheduling algorithm is its ***assigned priority***
  - At any time t, each ready job J_k is scheduled and executes at its ***current priority***, $\pi_k(t)$, 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
  - In such a situation, the lower-priority job J_l inherits the priority of the higher-priority job J_h, and J_l executes at its inherited priority $\pi_h(t)$

Resources and Resource Access Control

- Basic Priority-inheritance Protocol
  - 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. It remains at this priority except when the priority-inheritance rule is invoked.
  - Allocation Rule: when a job J requests a resource R at time t:
    - If R is free, R is allocated to J until J releases it
    - If R is not free, the request is denied and J is blocked
  - 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; J_l executes at its inherited priority 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 acquired the resource R

Resources and Resource Access Control

- What does this mean?
  - A job J is only denied a resource when the resource it requests is held by another job
  - At time t when J requests the resource, it has the highest priority among all ready jobs ➔ the current priority $\pi_l(t)$ of the job J_l directly blocking J is never higher than the priority $\pi(t)$ of J

Resources and Resource Access Control

- Consider the following system:

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

Resources and Resource Access Control

T	Ready Queue	Blocked Queue	red	blue	execute
0	J_5[5,6]	-	-	-	J_5
1	J_5[5,5]	-	-	J_5	J_5
2	J_4[4,6]; J_5[5,4]	-	-	J_5	J_4
3	J_4[4,5]; J_5[5,4]	-	J_4	J_5	J_4
4	J_3[3,2]; J_4[4,4]; J_5[5,4]	-	J_4	J_5	J_3
5	J_2[2,3]; J_3[3,1]; J_4[4,4]; J_5[5,4]	-	J_4	J_5	J_2
6	J_5[2,4]; J_3[3,1]; J_4[4,4]	J_2[2,2]	J_4	J_5	J_5
7	J_1[1,3]; J_5[2,3]; J_3[3,1]; J_4[4,4]	J_2[2,2]	J_4	J_5	J_1
8	J_4[1,4]; J_5[2,3]; J_3[3,1]	J_1[1,2]; J_2[2,2]	J_4	J_5	J_4
9	J_5[1,3]; J_3[3,1]	J_4[1,3]; J_1[1,2]; J_2[2,2]	J_4	J_5	J_5
11	J_4[1,3]; J_3[3,1]; J_5[5,1]	J_1[1,2]; J_2[2,2]	J_4	J_4	J_4
13	J_1[1,2]; J_2[2,2]; J_3[3,1]; J_4[4,1]; J_5[5,1]	-	J_1	J_2	J_1
14	J_1[1,1]; J_2[2,2]; J_3[3,1]; J_4[4,1]; J_5[5,1]	-	-	J_2	J_1
15	J_2[2,2]; J_3[3,1]; J_4[4,1]; J_5[5,1]	-	-	J_2	J_2
16	J_2[2,1]; J_3[3,1]; J_4[4,1]; J_5[5,1]	-	-	-	J_2
17	J_3[3,1]; J_4[4,1]; J_5[5,1]	-	-	-	J_3
18	J_4[4,1]; J_5[5,1]	-	-	-	J_4
19	J_5[5,1]	-	-	-	J_5
20	-	-	-	-	-

Resources and Resource Access Control

- Properties of the Priority-inheritance Protocol
  - Different types of blocking
    - Direct blocking (time 6)
    - Priority inheritance blocking (time 6)
    - Transitive blocking (time 9)
  - Does NOT prevent deadlock – simple piecemeal acquisition in different orders problem
  - Does not minimize the blocking times suffered by jobs since it is so aggressive

Resources and Resource Access Control

- Priority-ceiling Protocol
  - Extends priority-inheritance to prevent deadlock and to further reduce blocking times
  - Makes 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
  - Need two additional terms to define the protocol:
    - The ***priority ceiling*** of any resource R_k is the highest priority of all the jobs that require R_k and is denoted by $\Pi(R_k)$
    - At any time t, the current priority ceiling $\Pi(t)$ of the system is equal to the highest priority ceiling of the resources that are in use at the time; if all resources are free at the time, $\Pi(t)$ is equal to Ω, a nonexistent priority level that is lower than the lowest priority level of all jobs

Resources and Resource Access Control

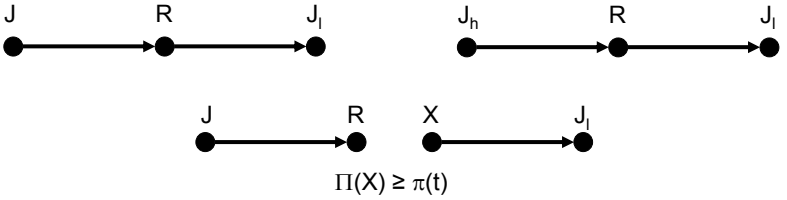
- The Basic Priority-Ceiling Protocol
  - Scheduling rules
    a. At its release time t, the current priority $\pi(t)$ of every job J is equal to its assigned priority; it remains at that priority except as defined in the priority-inheritance rule
    b. Every ready job J is scheduled preemptively and in a priority-driven manner at its current priority $\pi(t)$
  - Allocation rules – whenever a job J requests a resource R at time t, one of the following occurs:
    a. R is held by another job; J's request fails and J becomes blocked
    b. R is free
      i. If J's priority $\pi(t)$ is higher than the current priority ceiling $\Pi(t)$, R is allocated to J
      ii. If J's priority $\pi(t)$ is not higher than the ceiling $\Pi(t)$, R is allocated to J only if J is the job holding the resource(s) whose priority ceiling is equal to $\Pi(t)$; otherwise, J's request is denied, and J becomes blocked
  - 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)

Resources and Resource Access Control

T	Ready Queue	Blocked Queue	Π	red	blue	execute
0	J_5[5,6]	-	Ω	-	-	J_5
1	J_5[5,5]	-	2	-	J_5	J_5
2	J_4[4,6]; J_5[5,4]	-	2	-	J_5	J_4
3	J_5[4,4]	J_4[4,5]	2	-	J_5	J_5
4	J_3[3,2]; J_5[4,3]	J_4[4,5]	2	-	J_5	J_3
5	J_2[2,3]; J_3[3,1]; J_5[4,3]	J_4[4,5]	2	-	J_5	J_2
6	J_5[2,3]; J_3[3,1]	J_2[2,2]; J_4[4,5]	2	-	J_5	J_5
7	J_1[1,3]; J_5[2,2]; J_3[3,1]	J_2[2,2]; J_4[4,5]	2	-	J_5	J_1
8	J_1[1,2]; J_5[2,2]; J_3[3,1]	J_2[2,2]; J_4[4,5]	1	J_1	J_5	J_1
9	J_1[1,1]; J_5[2,2]; J_3[3,1]	J_2[2,2]; J_4[4,5]	2	-	J_5	J_1
10	J_5[2,2]; J_3[3,1]	J_2[2,2]; J_4[4,5]	2	-	J_5	J_5
11	J_2[2,2]; J_3[3,1]; J_5[5,1]	J_4[4,5]	2	-	J_2	J_2
12	J_2[2,1]; J_3[3,1]; J_5[5,1]	J_4[4,5]	Ω	-	-	J_2
13	J_3[3,1]; J_5[5,1]	J_4[4,5]	Ω	-	-	J_3
14	J_4[4,5]; J_5[5,1]	-	1	-	J_4	J_4
16	J_4[4,3]; J_5[5,1]	-	1	J_4	J_4	J_4
18	J_4[4,1]; J_5[5,1]	-	Ω	-	-	J_4
19	J_5[5,1]	-		-	-	J_5
20	-	-		-	-	-

Resources and Resource Access Control

- Differences between Priority-inheritance (PI) and Priority-ceiling (PC)
  - PI is greedy, while PC is not
  - Using PC, it is possible for a job J to be blocked by a lower-priority job which does not hold the requested resource – termed avoidance blocking
  - The priority ceilings forces a fixed order onto resource accesses, thus eliminating deadlock



Resources and Resource Access Control

- Important results for priority-ceiling
  - 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 (remember, this was for **fixed-priority** algorithms).
  - 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 – i.e. there is no transitive blocking under the priority-ceiling protocol.