completes. If transitive blocking were to occur, J_m would inherit $\pi_h(t)$, and J_l would inherit a priority higher than $\pi_m(t)$ indirectly. This leads to a contradiction. Hence, the supposition that the three jobs are transitively blocked must be false.

Computation of Blocking Time. Theorem 8.2 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.

To illustrate how to do this computation, let us consider the system of jobs whose resource requirements are given by Figure 8–14. As always, the jobs are indexed in order of decreasing priorities. We see that J_1 can be directed blocked by J_4 for 1 unit of time. The blocking time $b_1(rc)$ of J_1 is clearly one. 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 times $b_2(rc)$ and $b_3(rc)$ are also one.

Figure 8–15(a) shows a slightly more complicated example. Even for this small system, it is error prone if we compute the blocking times of all jobs by inspection, as we did earlier. The tables in Figure 8–15(b) give us a systematic way. There is a row for each job that can be blocked. (In these tables, there is a row for every job except J_6.) The tables list only the nonzero entries; all the other entries are zero. Since jobs are not blocked by higher-priority jobs, the entries at and below "*" in each column are zero.

The leftmost part is the direct-blocking table. It lists for each job the duration for which it can be directly blocked by each of the lower-priority jobs. The entries in this table come directly from the resource requirement graph of the system. Indeed, for the purpose of calculating the blocking times of the jobs, this table gives a complete specification of the resource requirements of the jobs.

The middle part of Figure 8–15(b) is the priority-inheritance blocking table. It lists the maximum duration for which each job can be priority-inheritance blocked by each of the lower-priority jobs. For example, J_6 can inherit priority π_1 of J_1 for 2 units of time when it directly blocks J_1. Hence, it can block all the other jobs for 2 units for time. In the table, we show 2 units of inheritance blocking time of J_2 and J_3 by J_6. However, because J_6 can also inherit π_3 for 4 units of time, it can block J_4 and J_5 for 4 units of time. This is the reason that the entries in the fourth and fifth rows of column 6 are 4. In general, a systematic way to get the entries in each column of this table from the entries in the corresponding column of the direct-blocking table is as follows. *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* $1, 2, \ldots, i-1$ *of the direct-blocking table.*

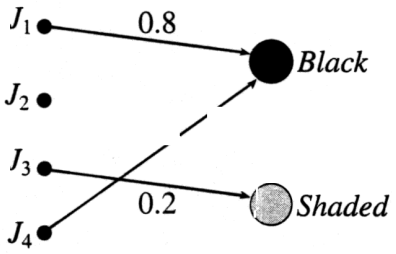
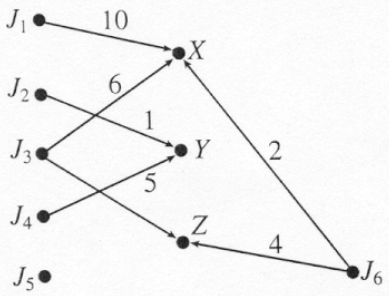


FIGURE 8–14 Example on duration of blocking.



(a)

	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				*	

(b)

FIGURE 8–15 Example illustrating the computation of blocking times.

The rightmost table in Figure 8–15(b) is the avoidance (priority-ceiling) blocking table. It lists the maximum duration for which each job can be avoidance blocked by each lower-priority job. Again, let us focus on column 6. When J_6 holds resource X (whose priority ceiling is the highest in the system), it avoidance blocks all the jobs which require any resource. Similarly, when it holds Z, it avoidance blocks J_4. Therefore, except for the entry in row 5, all entries in column 6 of the avoidance blocking table are the same as the corresponding entries in column 6 of the inheritance blocking table. J_5 does not require any resource and is never directly or avoidance blocked. In general, *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.* Jobs which do not require any resource are never avoidance blocked, just as they are never directly blocked.

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. From Figure 8–15(b), we have $b_i(rc)$ is equal to 6, 6, 5, 4, 4, and 0 for $i = 1, 2, \ldots, 6$, respectively

For this example, every entry in the avoidance blocking table is either equal to or smaller than the corresponding entries in the direct blocking or inheritance blocking tables. Since we

8.5.5

are taking the maximum value of the entries of each row, there is no need for us to compute the avoidance blocking table. Indeed, we do not need this table whenever the priorities of all the jobs are distinct. When the priorities of jobs are not distinct, a job may be avoidance blocked by a job of equal priority. The avoidance blocking table gives this information. For example, suppose that in addition to the jobs in Figure 8–15(a), there is a job J_1' whose priority is also π_1, and this job requires a resource V for 9 units of time. Then the blocking time of J_1' is 10, the amount of time J_1 holds the resource X and priority-ceiling blocks J_1'. Similarly, the blocking time of J_1 is 9, the duration for which it is priority-ceiling blocked by J_1'. In this case, we need the avoidance blocking table to give us these blocking times.

In Problem 8.14, you are asked to provide a pseudocode description of an algorithm that computes the blocking time $b_i(rc)$ of all the jobs from the resource requirement graph of the system. For the sake of efficiency, you may want to first identify for each job J_i the subset of all the jobs that may block the job. This subset is called the *blocking set* of J_i. (In our simple example, J_5 is not included in the blocking set of any other job, since it cannot block any job. The blocking set of J_i includes all the lower-priority jobs other than J_5.)

8.5.5 Fixed-Priority Scheduling and Priority-Ceiling Protocol

The priority-ceiling protocol is an excellent algorithm for controlling the accesses to resources of periodic tasks when the tasks are scheduled on a fixed-priority basis. It is reasonable to assume that the resources required by every job of every task and the maximum execution times of their critical sections are known a priori, just like the other task parameters. All the jobs in each periodic task have the same priority. Hence, the priority ceiling of each resource is the highest priority of all the tasks that require the resource. This makes it possible for us to analyze and determine the potential of resource contentions among tasks statically. The effect of resource contentions on the schedulability of the tasks can be taken care of by including the blocking time $b_i(rc)$ in the schedulability test of the system.

For example, suppose that the jobs in Figure 8–14 belong to four periodic tasks. The tasks are $T_1 = (\varepsilon, 2, 0.8; [Black; 0.8])$, $T_2 = (\varepsilon, 2.2, 0.4)$, $T_3 = (\varepsilon, 5, 0.2; [Shaded; 0.2])$, and $T_4 = (10, 1.0; [Black; 1.0])$, where ε is a small positive number. For all i, J_i in Figure 8–14 is a job in T_i. Figure 8–16 shows the initial segment of the schedule of the tasks according to the rate-monotonic algorithm and priority-ceiling protocol. We see that T_2 misses it is deadline at time $2.2 + \varepsilon$. A schedulability test can predict this miss. The time-demand function of T_2 is equal to 2.2 (i.e., $0.8 + 0.4 + 1.0$) in $(0, 2.0 + \varepsilon]$ when the blocking time $b_2(rc) = 1.0$ of T_2 is included and becomes 3.0 at $2.0 + \varepsilon$ (i.e., the beginning of the second period of T_1). Obviously, the time supply by T_2's first deadline at $2.2 + \varepsilon$ cannot meet this demand. Similarly, if the jobs J_i, for $i = 1, 2, \ldots, 6$, in Figure 8–15 are jobs in periodic tasks T_i, respectively, we can take the effect of resource conflicts into account in our determination of whether the tasks are schedulable by including the blocking time $b_i(rc)$ computed above in the schedulability test.

To summarize this section, we recall that two factors contribute to the time-demand function of each task in addition to the execution times of its jobs and execution times of equal and higher-priority jobs. They are blocking time and context-switching time. When the system is scheduled on a fixed-priority basis and uses the priority-ceiling protocol, we can compute the blocking time $b_i(rc)$ of each task T_i due to its resource conflicts with other tasks in the way described above. After we have thus obtained $b_i(rc)$, we include this blocking factor with the other types of blocking times (e.g., due to nonpreemptivity of lower-priority

EXERCISES

8. A system contains five jobs. There are three resources X, Y, and Z. The resource requirements of the jobs are listed below.

J_1: $[X; 2]$
J_2: none
J_3: $[Y; 1]$
J_4: $[X; 3\,[Z; 1]]$
J_5: $[Y; 4\,[Z; 2]]$

The priority J_i is higher than the priority of J_j for $i < j$. What are the maximum blocking times of the jobs under the nonpreemptable critical-section protocol and under the priority-ceiling protocol?

8.2 A system contains the following four periodic tasks. The tasks are scheduled by the rate-monotonic algorithm and the priority-ceiling protocol.

$T_1 = (3, 0.75)$ $b_1 = 0.9$
$T_2 = (3.5, 1.5)$ $b_2 = 0.75$
$T_3 = (6, 0.6)$ $b_3 = 1.0$
$T_4 = (10, 1)$

b_i is the blocking time of T_i. Are the tasks schedulable? Explain your answer.

8.3 Consider a fixed-priority system in which there are five tasks T_i, for $i = 1, 2, 3, 4$, and 5, with decreasing priorities. There are two resources X and Y. The critical sections of T_1, T_2, T_4, and T_5 are $[Y; 3]$, $[X; 4]$, $[Y; 5\,[X; 2]]$, and $[X; 10]$, respectively. (Note that T_3 does not require any resource.) Find the blocking times $b_i(rc)$ of the tasks.

8.4 A fixed-priority system contains four tasks T_i, for $i = 1, 2, 3, 4$, and 5, with decreasing priorities and uses the ceiling-priority protocol to control resource access. There are three resources X, Y, and Z; each has 1 unit. The critical sections of the tasks are $[X; 4]$, $[Y; 6]$, $[Z; 5]$, and $[X; 3\,[Y; 2\,[Z; 1]]]$, respectively. Suppose that T_2 may self-suspend once, and $b_2(ss)$ is 1. The other tasks never self-suspend. What are the blocking times of the tasks?

8.5 Sections 8.6.1 and 8.6.2 give two different implementations (and two different names) of the ceiling-priority protocol.

(a) Discuss the pros and cons of the implementations.

(b) The definitions of the stack-based, priority-ceiling protocol and ceiling-priority protocol do not say whether jobs are allowed to self-suspend. Do protocols still limit the duration of blocking if jobs may self-suspend? If yes, give an intuitive argument to support your answer. If no, give an illustrative example.

(c) Oftentimes, jobs of equal priority are scheduled on the round-robin basis. Modify the definition of priority ceilings of resources and the scheduling rule of the ceiling-priority protocol to make the protocol work for such jobs. (*Hint*: Consider restricting the priorities of all jobs to even integers. Define the priority ceiling of each resource to be the highest of the priorities of all jobs that require the resource minus one. In other words, the ceiling priorities of all resources are odd integers.)

8.6 A fixed-priority system contains five tasks. There are two kinds of resources X and Y. The resource X has 3 units and Y has 2 units. The resource requirements of the tasks are as follows: