In a multiprogramming environment, several threads may compete for a finite number of resources. A thread requests resources; if the resources are not available at that time, the thread enters a waiting state. Sometimes, a waiting thread is never again able to change state, because the resources it has requested are held by other waiting threads. This situation is called a *deadlock*. We discussed this issue briefly in Chapter 6 as a form of liveness failure. There, we defined deadlock as a situation in which *every process in a set of processes is waiting for an event that can be caused only by another process in the set.*

Perhaps the best illustration of a deadlock can be drawn from a law passed by the Kansas legislature early in the 20th century. It said, in part: “When two trains approach each other at a crossing, both shall come to a full stop and neither shall start up again until the other has gone.”

In this chapter, we describe methods that application developers as well as operating-system programmers can use to prevent or deal with deadlocks. Although some applications can identify programs that may deadlock, operating systems typically do not provide deadlock-prevention facilities, and it remains the responsibility of programmers to ensure that they design deadlock-free programs. Deadlock problems—as well as other liveness failures—are becoming more challenging as demand continues for increased concurrency and parallelism on multicore systems.

**Bibliographical Notes**

Most research involving deadlock was conducted many years ago. [Dijkstra (1965)] was one of the first and most influential contributors in the deadlock area. [Holt (1972)] was the first person to formalize the notion of deadlocks in terms of an allocation-graph model similar to the one presented in this chapter. Starvation was also covered by [Holt (1972)]. [Hyman (1985)] provided the deadlock example from the Kansas legislature. A study of deadlock handling is provided in [Levine (2003)].

The various prevention algorithms were suggested by [Havender (1968)], who devised the resource-ordering scheme for the IBM OS/360 system. The banker’s algorithm for avoiding deadlocks was developed for a single resource...
type by [Dijkstra (1965)] and was extended to multiple resource types by [Habermann (1969)].

The deadlock-detection algorithm for several instances of a resource type, which is described in Section 8.7.2, was presented by [Coffman et al. (1971)].

[Bach (1987)] describes how many of the algorithms in the traditional UNIX kernel handle deadlock. Solutions to deadlock problems in networks are discussed in works such as [Culler et al. (1998)] and [Rodeheffer and Schroeder (1991)].

Details for how the MySQL database manages deadlock can be found at http://dev.mysql.com/.

Bibliography


