7.8 Summary

A deadlocked state occurs when two or more processes are waiting indefinitely for an event that can be caused only by one of the waiting processes. There are three principal methods for dealing with deadlocks:

- Use some protocol to prevent or avoid deadlocks, ensuring that the system will never enter a deadlocked state.
- Allow the system to enter a deadlocked state, detect it, and then recover.
- Ignore the problem altogether and pretend that deadlocks never occur in the system.

The third solution is the one used by most operating systems, including UNIX and Windows.

A deadlock can occur only if four necessary conditions hold simultaneously in the system: mutual exclusion, hold and wait, no preemption, and circular wait. To prevent deadlocks, we can ensure that at least one of the necessary conditions never holds.

A method for avoiding deadlocks, rather than preventing them, requires that the operating system have a priori information about how each process that utilizes system resources. The banker’s algorithm, for example, requires a priori information about the maximum number of each resource class that each process may request. Using this information, we can define a deadlock-avoidance algorithm.

If a system does not employ a protocol to ensure that deadlocks will never occur, then a detection-and-recovery scheme may be employed. A deadlock-detection algorithm must be invoked to determine whether a deadlock has occurred. If a deadlock is detected, the system must recover either by terminating some of the deadlocked processes or by preempting resources from some of the deadlocked processes.

When preemption is used to deal with deadlocks, three issues must be addressed: selecting a victim, rollback, and starvation. In a system that selects victims for rollback primarily on the basis of cost factors, starvation may occur, and the selected process cannot complete its designated task.

Researchers have argued that none of the basic approaches alone is appropriate for the entire spectrum of resource-allocation problems in operating systems. The basic approaches can be combined, however, allowing us to select an optimal approach for each class of resources in a system.

Practice Exercises

7.1 List three examples of deadlocks that are not related to a computer-system environment.

7.2 Suppose that a system is in an unsafe state. Show that it is possible for the processes to complete their execution without entering a deadlocked state.

7.3 A possible method for preventing deadlocks is to have a single, higher-order resource that must be requested before any other resource. For example, if multiple threads attempt to access the synchronization objects A → E, deadlock is possible. (Such synchronization objects may include mutexes, semaphores, condition variables, and the like.) We can prevent the deadlock by adding a sixth object F. Whenever a thread wants to acquire the synchronization lock for any object A → E, it must first acquire the lock for object F. This solution is known as containment: the locks for objects A → E are contained within the lock for object F. Compare this scheme with the circular-wait scheme of Section 7.4.4.

7.4 Prove that the safety algorithm presented in Section 7.5.3 requires an order of \( n \times m^2 \) operations.

7.5 Consider a computer system that runs 5,000 jobs per month and has no deadlock-prevention or deadlock-avoidance scheme. Deadlocks occur about twice per month, and the operator must terminate and rerun about 10 jobs per deadlock. Each job is worth about $2 (in CPU time), and the jobs terminated tend to be about half-done when they are aborted.

A systems programmer has estimated that a deadlock-avoidance algorithm (like the banker’s algorithm) could be installed in the system with an increase in the average execution time per job of about 10 percent. Since the machine currently has 30 percent idle time, all 5,000 jobs per month could still be run, although turnaround time would increase by about 20 percent on average.

a. What are the arguments for installing the deadlock-avoidance algorithm?

b. What are the arguments against installing the deadlock-avoidance algorithm?

7.6 Can a system detect that some of its processes are starving? If you answer "yes," explain how it can. If you answer "no," explain how the system can deal with the starvation problem.

7.7 Consider the following resource-allocation policy. Requests for and releases of resources are allowed at any time. If a request for resources cannot be satisfied because the resources are not available, then we check any processes that are blocked waiting for resources. If a blocked process has the desired resources, then these resources are taken away from it and are given to the requesting process. The vector of resources for which the blocked process is waiting is increased to include the resources that were taken away.
Chapter 7  Deadlocks

For example, consider a system with three resource types and the vector Available initialized to (4,2,2). If process $P_1$ asks for (2,2,1), it gets them. If $P_2$ asks for (1,0,1), it gets them. Then, if $P_3$ asks for (0,1,1), it is blocked (resource not available). If $P_3$ now asks for (2,0,0), it gets the available one (1,0,0) and one that was allocated to $P_2$ (since $P_2$ is blocked). $P_3$'s Allocation vector goes down to (1,2,1), and its Need vector goes up to (1,0,1).

a. Can deadlock occur? If you answer "yes," give an example. If you answer "no," specify which necessary condition cannot occur.

b. Can indefinite blocking occur? Explain your answer.

7.8 Suppose that you have coded the deadlock-avoidance safety algorithm and now have been asked to implement the deadlock-detection algorithm. Can you do so by simply using the safety algorithm code and redefining Max = Waiting + Allocation, where Waiting is a vector specifying the resources for which process $i$ is waiting and Allocation is as defined in Section 7.5? Explain your answer.

7.9 Is it possible to have a deadlock involving only a single process? Explain your answer.

Exercises

7.10 Consider the traffic deadlock depicted in Figure 7.9.

a. Show that the four necessary conditions for deadlock hold in this example.

b. State a simple rule for avoiding deadlocks in this system.

7.11 Consider the traffic deadlock situation that can occur in the dining-philosophers problem when the philosophers obtain the chopsticks at a time. Discuss how the four necessary conditions for deadlock hold in this setting. Discuss how deadlocks could be avoided by eliminating any one of the four necessary conditions.

7.12 In Section 7.4.4, we describe a situation in which we prevent deadlock by ensuring that all locks are acquired in a certain order. However, we also point out that deadlock is possible in this situation if the threads simultaneously invoke the transaction() function. Fix the transaction() function to prevent deadlocks.

7.13 Compare the circular-wait scheme with the various deadlock-avoidance schemes (like the banker’s algorithm) with respect to the following issues:

a. Runtime overheads

b. System throughput

7.14 In a real computer system, neither the resources available nor the demands of processes for resources are consistent over long periods.

(m) Resources break or are replaced, new processes come and go, and new resources are bought and added to the system. If deadlock is controlled by the banker’s algorithm, which of the following changes can be made safely (without introducing the possibility of deadlock), and under what circumstances?

a. Increase Available (new resources added).

b. Decrease Available (resource permanently removed from system).

c. Increase Max for one process (the process needs or wants more resources than allowed).

d. Decrease Max for one process (the process decides it does not need that many resources).

e. Increase the number of processes.

f. Decrease the number of processes.

7.15 Consider a system consisting of four resources of the same type that are shared by three processes, each of which needs at most two resources. Show that the system is deadlock free.

7.16 Consider a system consisting of $m$ resources of the same type being shared by $n$ processes. A process can request or release only one resource at a time. Show that the system is deadlock free if the following two conditions hold:

a. The maximum need of each process is between one resource and $m$ resources.

b. The sum of all maximum needs is less than $m + n$. 
7.17 Consider the version of the dining-philosophers problem in which the chopsticks are placed at the center of the table and any two of them can be used by a philosopher. Assume that requests for chopsticks are made one at a time. Describe a simple rule for determining whether a particular request can be satisfied without causing deadlock given the current allocation of chopsticks to philosophers.

7.18 Consider again the setting in the preceding question. Assume now that each philosopher requires three chopsticks to eat. Resource requests are still issued one at a time. Describe some simple rules for determining whether a particular request can be satisfied without causing deadlock given the current allocation of chopsticks to philosophers.

7.19 We can obtain the banker’s algorithm for a single resource type from the general banker’s algorithm simply by reducing the dimensionality of the various arrays by 1. Show through an example that we cannot implement the multiple-resource-type banker’s scheme by applying the single-resource-type scheme to each resource type individually.

7.20 Consider the following snapshot of a system:

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Max Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C D</td>
<td>A B C D</td>
</tr>
<tr>
<td>P1</td>
<td>0 1 2</td>
</tr>
<tr>
<td>P2</td>
<td>1 0 3 4 5</td>
</tr>
<tr>
<td>P3</td>
<td>0 6 3 2</td>
</tr>
<tr>
<td>P4</td>
<td>0 0 4 1 6</td>
</tr>
</tbody>
</table>

Answer the following questions using the banker’s algorithm:

a. What is the content of the matrix Need?

b. Is the system in a safe state?

c. If a request from process P1 arrives for (0,4,2,0), can the requests be granted immediately?

7.21 What is the optimistic assumption made in the deadlock-detection algorithm? How can this assumption be violated?

7.22 A single-lane bridge connects the two Vermont villages of North Tunbridge and South Tunbridge. Farmers in the two villages use this bridge to deliver their produce to the neighboring town. The bridge can become deadlocked if a northbound and a southbound farmer get on the bridge at the same time (Vermont farmers are stubborn and are unable to back up.) Using semaphores, design an algorithm that prevents deadlock. Initially, do not be concerned about starvation (the situation in which northbound farmers prevent southbound farmers from using the bridge, or vice versa).

7.23 Modify your solution to Exercise 7.22 so that it is starvation-free.

Programming Problems

7.34 Write a multithreaded program that implements the banker’s algorithm discussed in Section 7.5.3. Create n threads that request and release resources from the bank. The banker will grant the request only if it leaves the system in a safe state. You may write this program using either Pthreads or Win32 threads. It is important that shared data be safe from concurrent access. To ensure safe access to shared data, you can use mutex locks, which are available in both the Pthreads and Win32 APIs. The use of mutex locks in both of these libraries is described in the project entitled “Producer–Consumer Problem” at the end of Chapter 6.

Wiley Plus

Visit Wiley Plus for
- Source code
- Solutions to practice exercises
- Additional programming problems and exercises
- Labs using an operating system simulator

Bibliographical Notes

Dijkstra [1965a] 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 recent 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 [1965a] and was extended to multiple resource types by Habermann [1969]. Exercises 7.15 and 7.16 are from Holt [1971].

The deadlock-detection algorithm for multiple instances of a resource type, which is described in Section 7.6.2, 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].

The witness lock-order verifier is presented in Baldwin [2002].