



Deadlocks

Practice Exercises

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

Answer:

- Two cars crossing a single-lane bridge from opposite directions.
- A person going down a ladder while another person is climbing up the ladder.
- Two trains traveling toward each other on the same track.
- Two carpenters who must pound nails. There is a single hammer and a single bucket of nails. Deadlock occurs if one carpenter has the hammer and the other carpenter has the nails.

- 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 deadlock state.

Answer: An unsafe state may not necessarily lead to deadlock, it just means that we cannot guarantee that deadlock will not occur. Thus, it is possible that a system in an unsafe state may still allow all processes to complete without deadlock occurring. Consider the situation where a system has 12 resources allocated among processes P_0 , P_1 , and P_2 . The resources are allocated according to the following policy:

	Max	Current	Need
P_0	10	5	5
P_1	4	2	2
P_2	9	3	6

```

for (int i = 0; i < n; i++) {
    // first find a thread that can finish
    for (int j = 0; j < n; j++) {
        if (!finish[j]) {
            boolean temp = true;
            for (int k = 0; k < m; k++) {
                if (need[j][k] > work[k])
                    temp = false;
            }

            if (temp) { // if this thread can finish
                finish[j] = true;
                for (int x = 0; x < m; x++)
                    work[x] += work[j][x];
            }
        }
    }
}

```

Figure 7.1 Banker's algorithm safety algorithm.

Currently there are two resources available. This system is in an unsafe state as process P_1 could complete, thereby freeing a total of four resources. But we cannot guarantee that processes P_0 and P_2 can complete. However, it is possible that a process may release resources before requesting any further. For example, process P_2 could release a resource, thereby increasing the total number of resources to five. This allows process P_0 to complete, which would free a total of nine resources, thereby allowing process P_2 to complete as well.

- 7.3** Prove that the safety algorithm presented in Section 7.5.3 requires an order of $m \times n^2$ operations.

Answer:

Figure 7.1 provides Java code that implement the safety algorithm of the banker's algorithm (the complete implementation of the banker's algorithm is available with the source code download).

As can be seen, the nested outer loops—both of which loop through n times—provide the n^2 performance. Within these outer loops are two sequential inner loops which loop m times. The big-oh of this algorithm is therefore $O(m \times n^2)$.

- 7.4** Consider a computer system that runs 5,000 jobs per month with 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?

Answer: An argument for installing deadlock avoidance in the system is that we could ensure deadlock would never occur. In addition, despite the increase in turnaround time, all 5,000 jobs could still run.

An argument against installing deadlock avoidance software is that deadlocks occur infrequently and they cost little when they do occur.

- 7.5 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.

Answer: Starvation is a difficult topic to define as it may mean different things for different systems. For the purposes of this question, we will define starvation as the situation whereby a process must wait beyond a reasonable period of time—perhaps indefinitely—before receiving a requested resource. One way of detecting starvation would be to first identify a period of time— T —that is considered unreasonable. When a process requests a resource, a timer is started. If the elapsed time exceeds T , then the process is considered to be starved.

One strategy for dealing with starvation would be to adopt a policy where resources are assigned only to the process that has been waiting the longest. For example, if process P_a has been waiting longer for resource X than process P_b , the request from process P_b would be deferred until process P_a 's request has been satisfied.

Another strategy would be less strict than what was just mentioned. In this scenario, a resource might be granted to a process that has waited less than another process, providing that the other process is not starving. However, if another process is considered to be starving, its request would be satisfied first.

- 7.6 Consider the following resource-allocation policy. Requests and releases for 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 they have the desired resources, then these resources are taken away from them and are given to the requesting process. The vector of resources for which the process is waiting is increased to include the resources that were taken away.

For example, consider a system with three resource types and the vector *Available* initialized to $(4,2,2)$. If process P_0 asks for $(2,2,1)$, it gets them. If P_1 asks for $(1,0,1)$, it gets them. Then, if P_0 asks for $(0,0,1)$, it is blocked (resource not available). If P_2 now asks for $(2,0,0)$, it gets the available one $(1,0,0)$ and one that was allocated to P_0 (since P_0 is blocked). P_0 '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.

Answer:

- a. Deadlock cannot occur because preemption exists.
- b. Yes. A process may never acquire all the resources it needs if they are continuously preempted by a series of requests such as those of process C.

7.7 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 $Max_i = Waiting_i + Allocation_i$, where $Waiting_i$ is a vector specifying the resources process i is waiting for, and $Allocation_i$ is as defined in Section 7.5? Explain your answer.

Answer:

Yes. The Max vector represents the maximum request a process may make. When calculating the safety algorithm we use the $Need$ matrix, which represents $Max - Allocation$. Another way to think of this is $Max = Need + Allocation$. According to the question, the $Waiting$ matrix fulfills a role similar to the $Need$ matrix, therefore $Max = Waiting + Allocation$.

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

Answer: No. This follows directly from the hold-and-wait condition.