Exercises

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

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

8.3 Consider the following snapshot of a system:

	Allocation	***Max***	***Available***
	A B C D	*A B C D*	*A B C D*
T_0	0 0 1 2	0 0 1 2	1 5 2 0
T_1	1 0 0 0	1 7 5 0	
T_2	1 3 5 4	2 3 5 6	
T_3	0 6 3 2	0 6 5 2	
T_4	0 0 1 4	0 6 5 6	

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 thread T_1 arrives for (0,4,2,0), can the request be granted immediately?

8.4 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 $A \cdots E$, deadlock is possible. (Such synchronization objects may include mutexes, semaphores, condition variables, and the like.) We can prevent deadlock by adding a sixth object F. Whenever a thread wants to acquire the synchronization lock for any object $A \cdots E$, it must first acquire the lock for object F. This solution is known as **containment**: the locks for objects $A \cdots E$ are contained within the lock for object F. Compare this scheme with the circular-wait scheme of Section 8.5.4.

8.5 Prove that the safety algorithm presented in Section 8.6.3 requires an order of $m \times n^2$ operations.

8.6 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 ten jobs per deadlock. Each job is worth about two dollars (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 of about 10 percent in the average execution time per job. 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?

8.7 Can a system detect that some of its threads are starving? If you answer “yes,” explain how it can. If you answer “no,” explain how the system can deal with the starvation problem.

8.8 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 threads that are blocked waiting for resources. If a blocked thread has the desired resources, then these resources are taken away from it and are given to the requesting thread. The vector of resources for which the blocked thread is waiting is increased to include the resources that were taken away.

For example, a system has three resource types, and the vector ***Available*** is initialized to (4,2,2). If thread T_0 asks for (2,2,1), it gets them. If T_1 asks for (1,0,1), it gets them. Then, if T_0 asks for (0,0,1), it is blocked (resource not available). If T_2 now asks for (2,0,0), it gets the available one (1,0,0), as well as one that was allocated to T_0 (since T_0 is blocked). T_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.

8.9 Consider the following snapshot of a system:

	Allocation	***Max***
	A B C D	*A B C D*
T_0	3 0 1 4	5 1 1 7
T_1	2 2 1 0	3 2 1 1
T_2	3 1 2 1	3 3 2 1
T_3	0 5 1 0	4 6 1 2
T_4	4 2 1 2	6 3 2 5

Using the banker's algorithm, determine whether or not each of the following states is unsafe. If the state is safe, illustrate the order in which the threads may complete. Otherwise, illustrate why the state is unsafe.

a. ***Available*** = (0,3,0,1)
b. ***Available*** = (1,0,0,2)

8.10 Suppose that you have coded the deadlock-avoidance safety algorithm that determines if a system is in a safe state or not, 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 for which thread *i* is waiting and ***Allocation***$_i$ is as defined in Section 8.6? Explain your answer.

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

8.12 In Section 6.8.1, we described a potential deadlock scenario involving processes P_0 and P_1 and semaphores `S` and `Q`. Draw the resource-allocation graph that illustrates deadlock under the scenario presented in that section.

8.13 Assume that a multithreaded application uses only reader–writer locks for synchronization. Applying the four necessary conditions for deadlock, is deadlock still possible if multiple reader–writer locks are used?

8.14 The program example shown in Figure E8.14 doesn't always lead to deadlock. Describe what role the CPU scheduler plays and how it can contribute to deadlock in this program.

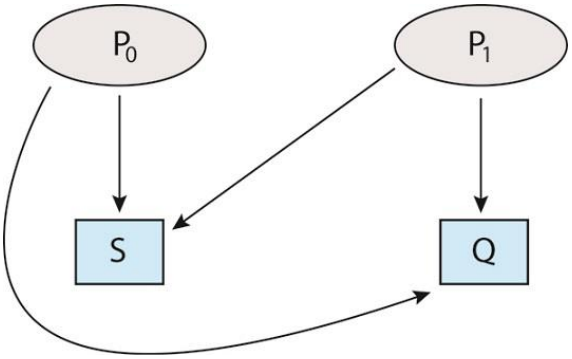

Figure E8.14 Resource-allocation graphs for Exercise 8.16.

8.15 In Section 8.5.4, we described 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 two threads simultaneously invoke the `transaction()` function. Fix the `transaction()` function to prevent deadlocks.

8.16 Which of the six resource-allocation graphs shown in Figure E8.15 illustrate deadlock? For those situations that are deadlocked, provide the cycle of threads and resources. Where there is not a deadlock situation, illustrate the order in which the threads may complete execution.

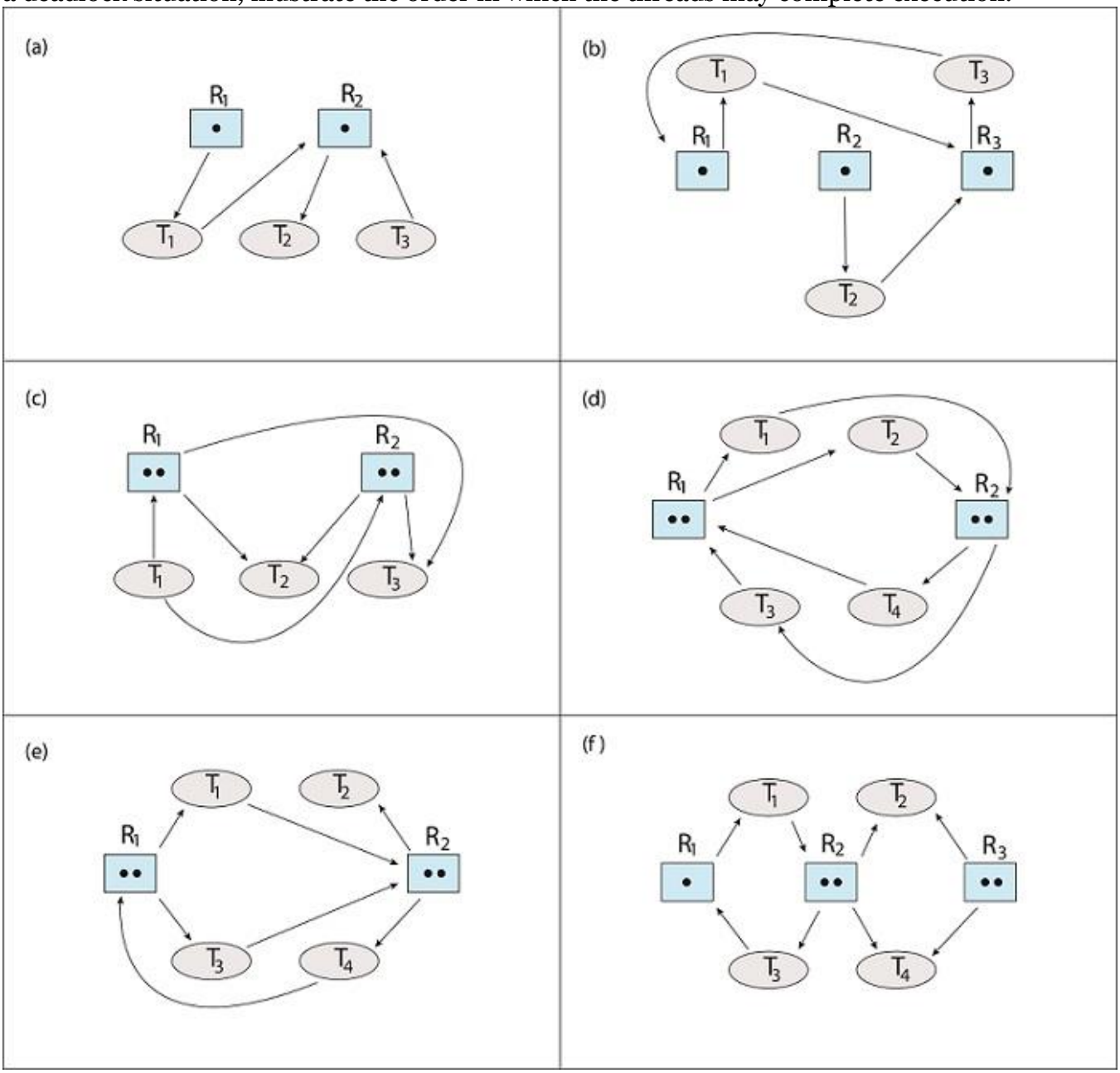

Figure E8.15 Resource-allocation graphs for Exercise 8.18.

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

a. Runtime overhead

b. System throughput

8.18 In a real computer system, neither the resources available nor the demands of threads for resources are consistent over long periods (months). Resources break or are replaced, new processes and threads 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 thread (the thread needs or wants more resources than allowed).
d. Decrease ***Max*** for one thread (the thread decides it does not need that many resources).

e. Increase the number of threads.

f. Decrease the number of threads.

8.19 Consider the following snapshot of a system:

	Allocation	***Max***
	A B C D	*A B C D*
T_0	2 1 0 6	6 3 2 7
T_1	3 3 1 3	5 4 1 5
T_2	2 3 1 2	6 6 1 4
T_3	1 2 3 4	4 3 4 5
T_4	3 0 3 0	7 2 6 1

What are the contents of the ***Need*** matrix?

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

8.21 Consider a system consisting of m resources of the same type being shared by n threads. A thread 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 thread is between one resource and m resources.
b. The sum of all maximum needs is less than $m + n$.

8.22 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.

8.23 Consider again the setting in the preceding exercise. 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.

8.24 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.

8.25 Consider the following snapshot of a system:

	Allocation	***Max***
	A B C D	*A B C D*
T_0	1 2 0 2	4 3 1 6
T_1	0 1 1 2	2 4 2 4
T_2	1 2 4 0	3 6 5 1
T_3	1 2 0 1	2 6 2 3
T_4	1 0 0 1	3 1 1 2

Using the banker's algorithm, determine whether or not each of the following states is unsafe. If the state is safe, illustrate the order in which the threads may complete. Otherwise, illustrate why the state is unsafe.

a. ***Available*** = (2,2,2,3)
b. ***Available*** = (4,4,1,1)
c. ***Available*** = (3,0,1,4)
d. ***Available*** = (1,5,2,2)

8.26 Consider the following snapshot of a system:

	Allocation	***Max***	***Available***
	A B C D	*A B C D*	*A B C D*
T_0	3 1 4 1	6 4 7 3	2 2 2 4
T_1	2 1 0 2	4 2 3 2	
T_2	2 4 1 3	2 5 3 3	
T_3	4 1 1 0	6 3 3 2	
T_4	2 2 2 1	5 6 7 5	

Answer the following questions using the banker's algorithm:

a. Illustrate that the system is in a safe state by demonstrating an order in which the threads may complete.

b. If a request from thread T_4 arrives for (2,2,2,4), can the request be granted immediately?
c. If a request from thread T_2 arrives for (0,1,1,0), can the request be granted immediately?
d. If a request from thread T_3 arrives for (2,2,1,2), can the request be granted immediately?

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

8.28 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 and/or mutex locks, design an algorithm in pseudocode 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).