SECR2043-07 OPERATING SYSTEMS

EXERCISE 7

Student's Name: Nurul Syamira binti Amat Jifri

Matric No.: A19EC0145

Lecturer's Name: Dr. Syed Hamid Hussain Madni

From Silberchatz, Operating System Concepts

Chapter 7 Exercises (page: 335 - 338)

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

1) Two persons running on a track in the stadium from opposite directions.

2) A person wants to enter through a door and another person also wants to exit through the same door.

3) Two trains moving towards each other on the same track from opposite directions.

7.6 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?

a) Ensure the deadlock will never occur.

b) Deadlock occur infrequently. And when when it occurs, it will cost only a little.

7.7 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.

No, this is because starvation needs out of box knowledgesince there is no features to keep records to determine it is happening or not. However, starvation can be prevented by 'aging' a process.

7.8 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 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.

For example, a system has three resource types, and the vector *Available* is 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), as well as 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.

a) No, deadlock cannot occur. This is because preemption exists.

b) Yes, indefinite blocking occur.

7.9 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 for which process *i* is waiting and $Allocation_i$ is as defined in Section 7.5? Explain your answer.

Yes. The *Max* vector defines the maximum request a process can make. There is an equation *Max = Need + Allocation*. For the question, the *Waiting* matrix is the same as *Need* matrix. Hence, the equation is *Max = Waiting + Allocation*.

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

No. This is because it is directly followed from the hold and wait condition.

7.12 Assume 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?

Yes, by applying the four necessary deadlock conditions which are mutual exclusion, hold and wait, no preemption and circular wait.

7.13 The program example shown in Figure 7.4 doesn't always lead to deadlock. Describe what role the CPU scheduler plays and how it can contribute to deadlock in this program.

In CPU, one process tries to allocate a resource that a second process holds.

7.14 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 two threads simultaneously invoke the transaction() function. Fix the transaction() function to prevent deadlocks.

To prevent deadlocks, make sure the locks cannot be acquired dynamically.