2013 - 2014 Thema 2.1 ITSD

Practicum Week 3 Solutions

Exercises

Chapter 7 Dead locks

Exercise 1.1 There are three major methods for handling deadlock. (See Chapter 7.3.1). But actually there are four. Describe these four methods for handling deadlock.

Answer 1.1

1. **Deadlock Prevention**: provides a set of methods for ensuring that at least one of the necessary conditions (Section 7.2.1) cannot hold. These method s prevent deadlocks by constraining how requests for resources can be made. We discuss these methods in Section 7.4.
2. **Deadlock Avoidance**: requires that the operating system be given in advance additional information concerning which resources a process will request and use during its lifetime. With this additional knowledge, it can decide for each request whether or not the process should wait. To decide whether the current request can be satisfied or must be delayed, the system must consider the resources currently available, the resources currently allocated to each process, and the future requests and releases of each process. We discuss these schemes in Section 7.5.
3. **Deadlock Detection**: If a system does not employ either a deadlock-prevention or a deadlock-avoidance algorithm, then a deadlock situation may arise. In this environment, the system can provide an algorithm that examines the state of the system to determine whether a deadlock has occurred and an algorithm to recover from the deadlock (if a deadlock has indeed occurred). We discuss these issues in Section 7.6 and Section 7.7.
4. **Deadlock Ignorance**: In the absence of algorithms to detect and recover from deadlocks, we may arrive at a situation in which the system is in a deadlocked state yet has no way of recognizing what has happened. In this case, the undetected deadlock will result in deterioration of the system’s performance, because resources are being held by processes that cannot run and because more and more processes, as they make requests for resources, will enter a deadlocked state. Eventually, the system will stop functioning and will need to be restarted manually.

Exercise 1.2 Consider the following snapshot of a system.

	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	1	5	2	0
P1	1	0	0	0	1	7	5	0				
P2	1	3	5	4	2	3	5	6				
P3	0	6	3	2	0	6	5	2				
P4	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 process Pi arrives for (0, 4, 2, 0), can the request be granted immediately?

Answer 1.2
a. What is the content of the matrix Need? The values of Need for processesP0 through P4 respectively are (0, 0, 0, 0), (0, 7, 5, 0), (1, 0, 0, 2), (0, 0, 2, 0), and (0, 6, 4, 2).

	Allocation				Max				Available				Need			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	1	5	2	0	0	0	0	0
P1	1	0	0	0	1	7	5	0					0	7	5	0
P2	1	3	5	4	2	3	5	6					1	0	0	2
P3	0	6	3	2	0	6	5	2					0	0	2	0
P4	0	0	1	4	0	6	5	6					0	6	4	2
	2	9	10	12	3	22	21	16	1	5	2	0	1	13	11	4

b. Is the system in a safe state? Yes. With Available being equal to (1,5, 2, 0), either process P0 or P3 could run. Once process P3 runs , it releases its resources, which allow all other existing processes to run.

c. If a request from process P1 arrives for (0, 4, 2, 0), can the request be granted immediately? Yes, it can. This results in the value of Available being (1, 1, 0, 0). One ordering of processes t hat can finish is P0, P2, P3, P1, and P4.

Exercise 1.3 Java's locking mechanism (the synchronized statement) is considered reentrant. That is, if a thread acquires the lock for an object (by invoking a synchronized method or block), i t can enter other synchronized methods or blocks for the same object. Explain how deadlock would be possible i f Java's locking mechanism were not reentrant.

Answer 1.3 A thread could simply deadlock itself by first acquiring an object lock (say v i a a synchronized method) and then attempting to acquire t he lock again (perhaps i n another synchronized method or block.) T his second attempt would fail if the lock were not reentrant.

Programming Exercise

Exercise 2.1 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 Bibliographical Notes 271 either Pthreads or Win32 threads. It is important that access to shared data is safe from concurrent access. Such data can be safely accessed using mutex locks, which are available in both the Pthreads and Win32 API. Coverage of mutex locks in both of these libraries is described in "'producer-consumer problem" project in Chapter 6.

Answer 2.1 See Banker-solution.zip