Operating Systems - 2 Spring 2022
Theory Assignment 2 - Deadlock Exercises

Question 8.22

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.

Solution

Here, we prove it by contradiction. Here, for the system to be deadlocked, they should be in circular wait for being in a deadlock state. Here, it is not possible because it is only possible if every thread is holding one process and waiting for another thread at the same time. Here, there are 4 resources and 3 threads, So, at least one thread should obtain two resources i.e, the deadlock has not occurred because after the thread completes the execution it returns the resources to other threads to execute.

Question 8.24

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.

Solution

Here, deadlock occurs when all the philosophers are stuck with one chopstick and then every philosopher has to wait for the next chopstick and there are no chopsticks left. Here, to solve the deadlock problem, we can stop processing the request of a philosopher for the first chopstick if there is only one chopstick left on the table and there is no philosopher with two chopsticks with him. Here, this step prevents the deadlock from occurring.

Question 8.28

A. Here, the safe state order is T2,T3,T0,T1,T4

	Allocation	Max	Available	Need
	A B C D	A B C D	A B C D	A B C D
T0	3 1 4 1	6 4 7 7	2 2 2 4	3 3 3 6
T1	2 1 0 2	4 2 3 2	4 6 3 7	2 1 3 0
T2	2 4 1 3	2 5 3 3	8 7 4 7	0 1 2 0
T3	4 1 1 0	6 3 3 2	11 8 8 8	2 2 2 2
T4	2 2 2 1	5 6 7 5	13 9 8 10	3 4 5 4
	13 9 8 7		15 11 10 11	

Here, the need of T2 is (0,1,2,0), which can be allocated by the system at first. So, it will execute first. Next, after the T2 process is executed, the resources which were allocated to T2 will be taken by the system, Now, the system has (4,6,3,7) resources, Now, the need of T3 is (2,2,2,2), which can be allocated by the system now. So, it will execute T3 now. Next, after the T3 process is executed, the resources which were allocated to T3 will be taken by the system, Now, the system has (8,7,4,7) resources, Now, the need of T0 is (3,3,3,6), which can be allocated by the system now. So, it will execute T0 now. Next, after the T0 process is executed, the resources which were allocated to T0 will be taken by the system, Now, the system has (11,8,8,8) resources, Now, the need of T1 is (2,1,3,0), which can be allocated by the system now. So, it will execute T1 now. Next, after the T1 process is executed, the resources which were allocated to T1 will be taken by the system, Now, the system has (13,9,8,10) resources, Now, the need of T4 is (3,4,5,4), which can be allocated by the system now. So, it will execute T4 now. Next, after the T4 process is executed, the resources which were allocated to T4 will be taken by the system. Now, the system has (15,11,10,11) resources, Now, all the processes are executed properly and there are no deadlocks.

B. Here, a request from thread T4 arrives for (2,2,2,4), if the request is granted immediately, then the available resources will be (0,0,0,0), now, no process can execute because there are no resources available for the need of any process. So, it is not possible to grant the request immediately.

C. Here, a request from thread T2 arrives for (0,1,1,0), it can be granted immediately because if it is granted still, the processes can execute in the order T2,T3,T0,T1,T4 with the system in a safe state throughout the execution.

	Allocation	Max	Available	Need
	A B C D	A B C D	A B C D	A B C D
T0	3 1 4 1	6 4 7 7	2 1 1 4	3 3 3 6
T1	2 1 0 2	4 2 3 2	4 6 3 7	2 1 3 0
T2	2 5 2 3	2 5 3 3	8 7 4 7	0 0 1 0
T3	4 1 1 0	6 3 3 2	11 8 8 8	2 2 2 2
T4	2 2 2 1	5 6 7 5	13 9 8 10	3 4 5 4
	13 9 8 7		15 11 10 11	

D. Here, a request from thread T3 arrives for (2,2,1,2), it can be granted immediately because if it is granted still, the process can execute in the order T3,T1,T2,T0,T4 with the system in a safe state throughout the execution.

	Allocation	Max	Available	Need
	A B C D	A B C D	A B C D	A B C D
T0	3 1 4 1	6 4 7 7	0 0 1 2	3 3 3 6
T1	2 1 0 2	4 2 3 2	6 3 3 4	2 1 3 0
T2	2 4 1 3	2 5 3 3	8 4 3 6	0 1 2 0
T3	6 3 2 2	6 3 3 2	10 8 4 9	0 0 1 0
T4	2 2 2 1	5 6 7 5	13 9 8 10	3 4 5 4
	15 11 9 9		15 11 10 11	

Question 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?

Solution

A. If we install a deadlock-avoidance algorithm, it makes sure that there are no deadlocks occurring in the process, So, we don't have to rerun 10 processes each time a deadlock occurs. It will ensure the smooth running of processes without any deadlocks. Here, even if the waiting time of the processes increases, we can clearly see that all the processes can still run.
B. Here, we can see that the deadlocks don't occur frequently but they occur very rarely which is around twice per month. Here, even the cost of removing a deadlock or rerunning the processes again is very little. So, we even can see that without the deadlock-avoidance algorithm, it will even decrease the turnaround time.

Variation of Question 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 ⋯ 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 ⋯ 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. Is there any drawback of this scheme?

Solution

Here, consider a case where there are 3 threads, and the threads need the resources
1st -> A
2nd -> B,C
3rd -> D,E
Here, the process takes place in the following way: first, the 1st thread locks the resources and uses only A, next the second thread locks the resources and uses only B,C, then the third thread locks the resources and uses only D,E. Here, we can see that the 3 threads can run simultaneously as a multithreaded process but due to

containment, we cannot run them simultaneously as a multithreaded process. So, here it increases the waiting time of the processes which can be done much faster without containment.