Question 1

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

A critical drawback of this scheme is that if the number of threads is large, then the waiting time for acquiring the lock on the higher-order resource (resource F) can be very high which can thus affect the performance of the program.

Another drawback is that it will increase the complexity of the code for the developer and it is up to the developer to ensure that the code follows the protocol of acquiring the higher-order resource first.

Thus the performance and complexity of the program can be affected by the containment scheme.

Question 2

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 T0 asks for (2,2,1), it gets them. If T1 asks for (1,0,1), it gets them. Then, if T0 asks for (0,0,1), it is blocked (resource not available). If T2 now asks for (2,0,0), it gets the available one (1,0,0), as well as one that was allocated to T0 (since T0 is blocked). T0 '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.

Solution

No, deadlock cannot occur in the above resource allocation policy. This is because the “No preemption” condition necessary for deadlock is violated here. A thread can preempt other blocked threads to release the resources that it requires. Thus the resource allocation policy is preemptive and thus deadlock will not occur.

Yes, indefinite blocking can occur. This can be seen clearly in the case of the dining philosophers problem. For example, it can happen that philosopher 2 starts eating every time before philosopher 5 stops eating and philosopher 5 starts eating every time before philosopher 2 stops eating. In this example, philosopher 1 is indefinitely blocked. Philosopher 1 never acquires all the resources it requires because it keeps getting preempted by philosophers 2 and 5.

Question 3

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

Suppose that the system is in a deadlock. Then the circular waiting condition must be satisfied. There should be at least two processes waiting for resources as a cycle cannot be formed otherwise. This can happen in two cases,

a. When two threads have one resource each and the third thread has 2 resources
b. When each thread has one resource

In case (a), the thread that has 2 resources will eventually complete and release the resources. Then the waiting threads will acquire the resources and complete their execution. Thus we do not have a deadlock.

In case (b), 1 resource is still available as we have a total of 4 resources. So either thread will acquire that resource and the state will transition to case (a) in which deadlock is not possible. Thus a deadlock is not possible in this case either.

Thus the system is deadlock free.

Question 4

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. (a) Describe a simple rule for determining whether a particular request can be satisfied without causing deadlock given the current allocation of chopsticks to philosophers. (b) Explain the practical difficulty in implementing this rule.

Solution

Simple rule: Allocate chopsticks two at a time.

Using the above rule, the “hold and wait” condition for deadlock is invalidated and thus deadlock cannot happen.

Practical difficulty: The above scheme can be implemented by having a variable (say “A”) which records the number of available chopsticks. “A” should be protected to avoid race conditions. This can be done using a mutex lock.

Ideally, we cannot have a situation where any philosopher has only one chopstick as we allocate only two chopsticks at a time.

We must ensure that a philosopher attempts to pick up only two chopsticks at a time, not one. To pick up chopsticks, the philosopher should acquire the lock on “A” first. If $A>1$, reduce A by 2, then release the lock. This should be implemented by the philosopher which may increase complexity of code.

Question 5

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?

Solution

a. Order for completion: T2, T3, T1, T0, T4
b. No, this request cannot be granted as the system will go into an unsafe state
c. Yes, this request can be granted. The system will still be in safe state. The order for completion will be: T2, T3, T1, T0, T4
d. Yes, this request can be granted. The system will still be in safe state. The order for completion will be: T3, T2, T1, T0, T4