# Operating Systems 2

## Tanmay Garg CS20BTECH11063

Theory Assignment 2

**Q1. 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?**

Sol.

* The time for running processes will increase significantly
* It might be better to use a locking mechanism which doesn’t significantly increase the runtime for the processes
* In comparison with circular-wait algorithm, it is a better approach to avoid deadlock and also does not increase runtime for each process

**Q2. 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?**

Sol.

A.

* We can ensure that deadlock doesn’t occur at all
* With the avoidance algorithm, the execution would be more reliable as there won’t be any jobs that would be incomplete and have to be rerun
* There won’t be any need of the operator, as now the system will be self sufficient
* Even though there is an increase in turnaround time, it is claimed that all the jobs can be run without any issues

B.

* Deadlocks do not occur very frequently, and it is always 10 jobs for every 5000 jobs
* All the processes will now run slowly due to the avoidance system
* Even if the deadlocks occur, the cost to rerun the those processes is still less than using the avoidance algorithm

**Q3. 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.**

Sol.

* Let us take a case when the system is in a deadlock condition
* Each process is holding one resource and is waiting for another
* Now, there are 3 processes and 4 resources, a single process can acquire 2 resources and complete its job
* After completion it won’t require the resources and will release them
* Hence, this system is deadlock free system

**Q4. 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**

**a particular request can be satisfied without causing deadlock given the current allocation of**

**chopsticks to philosophers.**

Sol.

* The request for taking the first chopstick must be rejected when there is no other philosopher with 2 chopsticks and when there is only 1 chopstick remaining
* This won’t cause the system to be in a deadlock situation because the philosopher will not get the chopstick as there is only 1 chopstick remaining, so it can be used by some other philosopher who currently has one chopstick and wants one more to complete the task

**Q5. Consider the following snapshot of a system:**

**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 T4 arrives for (2, 2, 2, 4), can the request be granted**

**immediately?**

**C. If a request from thread T2 arrives for (0, 1, 1, 0), can the request be granted**

**immediately?**

**D. If a request from thread T3 arrives for (2, 2, 1, 2), can the request be granted**

**Immediately**

Sol.

A.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Allocation | | | | Max | | | | Need | | | |
|  | A | B | C | D | A | B | C | D | A | B | C | D |
| T0 | 3 | 1 | 4 | 1 | 6 | 4 | 7 | 7 | 3 | 3 | 3 | 6 |
| T1 | 2 | 1 | 0 | 2 | 4 | 2 | 3 | 2 | 2 | 1 | 3 | 0 |
| T2 | 2 | 4 | 1 | 3 | 2 | 5 | 3 | 3 | 0 | 1 | 2 | 0 |
| T3 | 4 | 1 | 1 | 0 | 6 | 3 | 3 | 2 | 2 | 2 | 2 | 2 |
| T4 | 2 | 2 | 2 | 1 | 5 | 6 | 7 | 5 | 3 | 4 | 5 | 4 |

* For T0: Need[i] < Available is false
* For T1: Need[i] < Available is false
* For T2: Need[i] < Available is true
  + Finish = [F, F, T, F, F]
  + Available = 4 6 3 7
* For T0: Need[i] < Available is true
  + Finish = [T, F, T, F, F]
  + Available = 7 7 7 8
* For T1: Need[i] < Available is True
  + Finish = [T, T, T, F, F]
  + Available = 9 8 7 10
* For T3: Need[i] < Available is True
  + Finish = [T, T, T, T, F]
  + Available = 13 9 8 10
* For T4: Need[i] < Available is True
  + Finish = [T, T, T, T, T]
* So the order of running will be T2, T0, T1, T3, T4
* The system is in a safe state

B.

* The available matrix becomes [0 0 0 0]
* No thread can run
* The system goes into unsafe state

C.

* The available matrix becomes [2 1 1 4]

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Allocation | | | | Max | | | | Need | | | |
|  | A | B | C | D | A | B | C | D | A | B | C | D |
| T0 | 3 | 1 | 4 | 1 | 6 | 4 | 7 | 7 | 3 | 3 | 3 | 6 |
| T1 | 2 | 1 | 0 | 2 | 4 | 2 | 3 | 2 | 2 | 1 | 3 | 0 |
| T2 | 2 | 5 | 2 | 3 | 2 | 5 | 3 | 3 | 0 | 0 | 1 | 0 |
| T3 | 4 | 1 | 1 | 0 | 6 | 3 | 3 | 2 | 2 | 2 | 2 | 2 |
| T4 | 2 | 2 | 2 | 1 | 5 | 6 | 7 | 5 | 3 | 4 | 5 | 4 |

* T2 process can run
* So the order of running will be T2, T0, T1, T3, T4
* The system is in a safe state

D.

* The available matrix becomes [0 0 1 2]

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Allocation | | | | Max | | | | Need | | | |
|  | A | B | C | D | A | B | C | D | A | B | C | D |
| T0 | 3 | 1 | 4 | 1 | 6 | 4 | 7 | 7 | 3 | 3 | 3 | 6 |
| T1 | 2 | 1 | 0 | 2 | 4 | 2 | 3 | 2 | 2 | 1 | 3 | 0 |
| T2 | 2 | 4 | 1 | 3 | 2 | 5 | 3 | 3 | 0 | 1 | 2 | 0 |
| T3 | 6 | 3 | 2 | 2 | 6 | 3 | 3 | 2 | 0 | 0 | 1 | 0 |
| T4 | 2 | 2 | 2 | 1 | 5 | 6 | 7 | 5 | 3 | 4 | 5 | 4 |

* For T3: Need[i] < Available
  + Available = [6 3 3 4]
* For T1: Need[i] < Available
  + Available = [6 4 3 6]
* For T0: Need[i] < Available
  + Available = [11 5 7 7]
* So the order of running will be T3, T1, T0, T2, T4
* The system is in a safe state