CS3523: Operating Systems - II

Theory Assignment 2

# Soumi Chakraborty

ES19BTECH11017

# Variant of Question 8.4

The drawback of the proposed solution is that it is only beneficial in scenarios when there may be a deadlock. Even when there is no deadlock, this scheme will allow only one process access to one of the resources while the other resources do nothing when they could have been being used by other processes to maximise productivity.

# Question 8.6

## A. For

Installing the deadlock-avoidance algorithm would mean that no one has to manually rerun the half-completed jobs every month, and removing deadlocks would save $20 (2 \* 10 \* 1) of CPU time. It also has the obvious advantage of no deadlocks ever.

## B. Against

Installing the deadlock-avoidance algorithm would result in a much higher turnaround time which means a larger cost in CPU time. Thus one would be wasting much more money and time just to save $20 to prevent occasional manual labour and deadlocks.

# Question 8.22

A deadlock will only arise when a process has an insufficient amount of resources and is waiting to get more. In the given example, one process requires 2 resources at the most. Let’s consider the following cases:

1. **Each process needs only one resource**

This clearly is a deadlock-free system since every process can acquire one resource, finish their task while one resource is left completely free the entire time.

1. **One process requires two resources, the others one**

Even this is a deadlock-free situation since there are four resources and one thread can use two while leaving the other two for the remaining two threads

1. **Two processes require two resources**

In this scenario, if two processes use two resources at the same time, the third thread will need to wait. However, since both threads have sufficient resources, they are bound to finish their task and free them for the waiting thread.

Thus, we can argue the same way for all possible scenarios to show that this system is deadlock-free.

# Question 8.24

The only possible scenario for a deadlock in the dining-philosophers problem arises when each philosopher has only one chopstick, and are simultaneously waiting for the second chopstick. To prevent this from happening, we can allow a philosopher with no chopstick to only pick up a chopstick when there are at least two chopsticks available. This way, other philosophers with only one chopstick can pick up the last chopstick, finish eating, return them, after which the waiting philosopher can pick them up.

# Question 8.28

|  | **Allocation** | **Max** | **Available** | **Needs** |
| --- | --- | --- | --- | --- |
|  | **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 |  | 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 |

# A. The table below shows that the system will remain in a safe state for the following order of execution. Please note that there may be other orders of execution as well.

|  | **Work** | **Finish** | **Needs** | **Available (after completion)** |
| --- | --- | --- | --- | --- |
|  | **A B C D** | **A B C D** | **A B C D** | **A B C D** |
| **T2** | 2 2 2 4 | F F F F F | 0 1 2 0 | 4 6 3 7 |
| **T3** | 4 6 3 7 | F F T F F | 2 2 2 2 | 8 7 4 7 |
| **T1** | 8 7 4 7 | F F T T F | 2 1 3 0 | 10 8 4 9 |
| **T0** | 10 8 4 9 | F T T T F | 3 3 3 6 | 13 9 8 10 |
| **T4** | 13 9 8 10 | T T T T F | 3 4 5 4 | 15 11 10 11 |
|  |  | T T T T T |  |  |

# B. New table.

|  | **Allocation** | **Max** | **Available** | **Needs** |
| --- | --- | --- | --- | --- |
|  | **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 0 0** | 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** | **4 4 4 5** | **5 6 7 5** |  | **1 2 3 0** |

# Since available resources are all zero, this request cannot be granted.

# C. New table.

|  | **Allocation** | **Max** | **Available** | **Needs** |
| --- | --- | --- | --- | --- |
|  | **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 |  | 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 |

# This request can be granted since 0 0 1 0 is needed while 2 1 1 4 is available. After T2 completes, the system will continue in a safe state.

# D. New table.

|  | **Allocation** | **Max** | **Available** | **Needs** |
| --- | --- | --- | --- | --- |
|  | **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 |  | 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 |

# This request can be granted since 0 0 1 0 is needed while 0 0 1 2 is available. After T3 completes, the system will continue in a safe state.