CS3523: Operating Systems - II

Quiz 3

# Soumi Chakraborty

ES19BTECH11017

# Question 1

# (a)

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

# 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** |
| **P0** | 1 5 2 0 | F F F F F | 0 0 0 0 | 1 5 3 2 |
| **P3** | 1 5 3 2 | T F F F F | 0 0 2 0 | 1 11 6 4 |
| **P2** | 1 11 6 4 | T F F T F | 1 0 0 2 | 2 14 11 8 |
| **P1** | 2 14 11 8 | T F T T F | 0 7 5 0 | 3 14 11 8 |
| **P4** | 3 14 11 8 | T T T T F | 0 6 4 2 | 3 14 12 12 |
|  |  | T T T T T |  |  |

# Thus, using the algorithm, we’ve proven that the state is safe. The safe sequence is: P0, P3, P3, P1, P4.

# (b)

|  | **Allocation** | **Max** | **Available** | **Needs** |
| --- | --- | --- | --- | --- |
|  | **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 2 |  | 0 7 5 2 |
| **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 |

# New table. Assuming that the requested resources are in addition to the already allocated resources.

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

This cannot be granted immediately since the number of available resources is lesser than those that P1 needs. Granting this request would send the program into an unsafe state, hence this isn’t granted immediately.

# 

# Question 2

**Explain how the safety algorithm discussed in the class requires an order of m×n**2

**operations.**

# Here, m = number available resources of each type and n = number of processes

# The algorithm detects every possible sequence of allocation of resources to processes. To explain the order of operations, we can try to code this algorithm with the help of loops. For this algorithm, we would need 3 loops, one for the resource types (O(m)), and two for iterating through all combinations of the n processes (O(n2)). Thus, when combined together, the order of operations for the algorithm is O(m**×**n2)).

# Question 3

**Suppose you wish to implement Banker’s algorithm for Deadlock Avoidance.**

**(a) Please explain where will you store the data structures required by the**

**algorithm: Max, Allocation, Need.**

The data structures will be stored like so:

* Max: globally
* Allocation: globally
* Need: globally

They are all global since all these data structures are required by the processes at all times, and have the same value for each process.

**(b) How processes will modify these data structures to ensure the correct working**

**of the algorithm? What are the techniques you will use to protect the data structure**

**used by the algorithm and ensure correctness?**

To ensure correctness, we need to synchronize the modification requests properly. To do so, we can make use of semaphores. Let this semaphore be initialised to 1 and be called lock. The max data structure doesn’t need to be modified since that is a constant that cannot change.

For allocation and need, we can use wait(lock), update as needed and then use signal(lock) to indicate that the modification is complete.

# Question 4

**Consider a system with p processes each needing a maximum of m resources and a**

**total of r resources available. What condition must hold to make the system deadlock free?**

A deadlock arises when processes have some resources but need more resources than what is currently available, so they wait until they have all the resources they need. Thus, let’s consider a worst-case scenario where each process has all the resources it needs, except for one. That is, each process has m - 1 resources.

This makes the total number of allocated resources p \* (m - 1). To keep this system deadlock-free, this number needs to be strictly lesser than the total number of available resources, and not lesser than equal to. If p \* (m - 1) was equal to r, then that means there are no free resources for even one of the processes, which means that the system would remain in a deadlocked state because all the processes would continue waiting indefinitely.

This can be written in the following ways:

**p \* (m - 1) < r**

OR

**p \* (m - 1) <= r - 1**

# Question 5

**Can you modify the transaction function so that it does not cause deadlocks?**

The following changes prevent the system from dynamically acquiring locks to prevent deadlocks.

void transaction(Account from, Account to, double amount)

{

semaphore lock1 = 1, lock2 = 1;

lock1 = get\_lock(from);

lock2 = get\_lock(to);

wait(lock1);

withdraw(from, amount);

signal(lock1);

wait(lock2);

deposit(to, amount);

signal(lock2);

}