Romil V. Shah
20008692

CS 520
Prof. Igor Faynberg

Homework Assignment 5
Submission Date: 11 / 01 / 2022

Q1. No. 8.11.2

(a) Suppose that a system is in an unsafe state. Show that it is possible for the threads to complete their execution without entering a deadlocked state.

Sol.

If Operating System is not able to prevent processes from requesting resources which can also lead to deadlock, then the system is said to be in an unsafe state. Unsafe State does not necessarily cause deadlock it may or may not cause deadlock.

When the deadlock happens, the processes that are waiting to acquire resources are held by another process. None of the job completes execution and it suffers from deadlock. The system turns into safe state, and it uses the released resources of job to complete the execution of other jobs. It is a reason why the system in an unsafe state is not necessarily deadlocked.

An unsafe state may not necessarily lead to deadlock, it just means that we cannot guarantee that deadlock will not occur. Thus, it is possible that a system in an unsafe state may still allow all processes to complete without deadlock occurring.

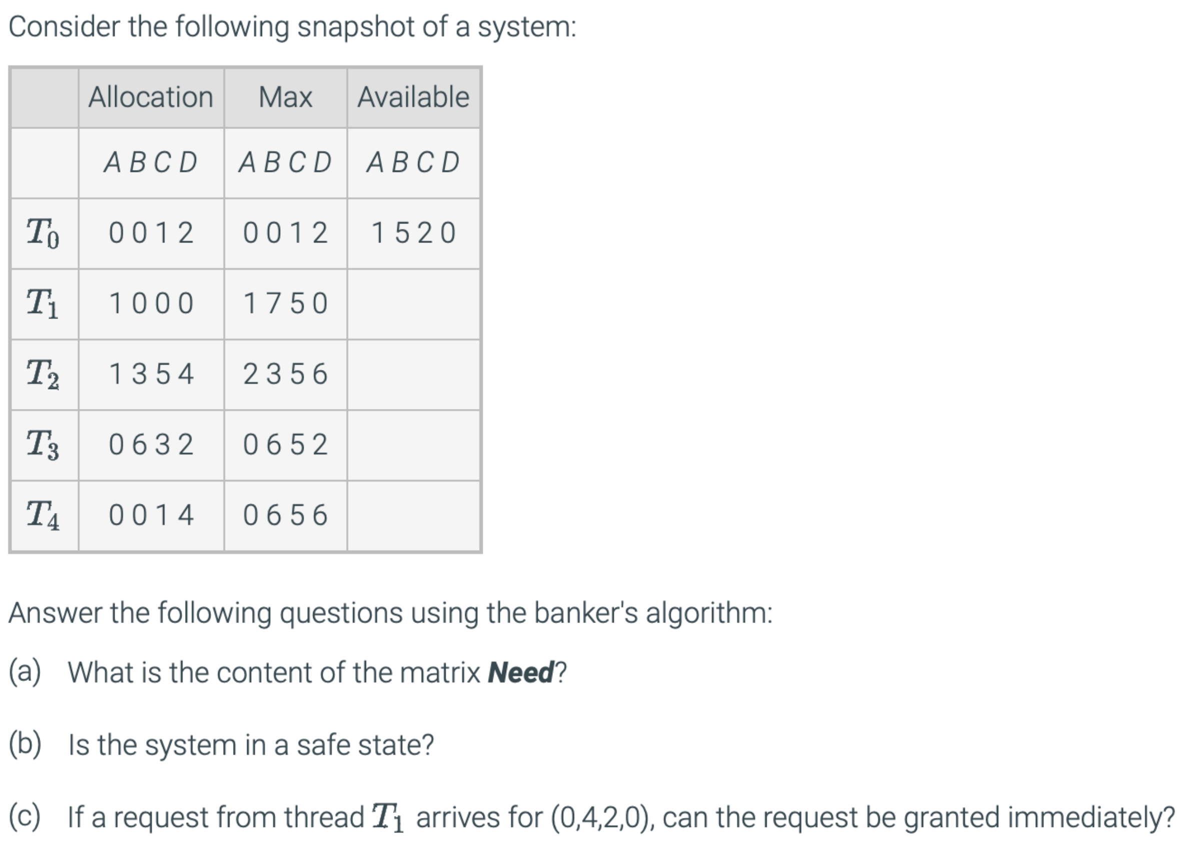
Let us consider the following situation where a system has 12 resources allocated among processes P0, P1, and P2. The resources are allocated according to the following policy:

	Max	Current	Need
P0	10	5	5
P1	4	2	2
P2	9	3	6

Currently there are two resources available. This system is in an unsafe state as process P1 could complete, thereby freeing a total of four resources. But we cannot guarantee that processes P0 and P2 can complete.

However, it is possible that a process may release resources before requesting any further. For example, process P2 could release a resource, thereby increasing the total number of resources to five. This allows process P0 to complete, which would free a total of nine resources, thereby allowing process P2 to complete as well.

Q2. No. 8.11.3



Consider the following snapshot of a system:

	Allocation	Max	Available
	A B C D	*A B C D*	*A B C D*
T_0	0 0 1 2	0 0 1 2	1 5 2 0
T_1	1 0 0 0	1 7 5 0	
T_2	1 3 5 4	2 3 5 6	
T_3	0 6 3 2	0 6 5 2	
T_4	0 0 1 4	0 6 5 6	

Answer the following questions using the banker's algorithm:

(a) What is the content of the matrix ***Need***?

(b) Is the system in a safe state?

(c) If a request from thread T_1 arrives for (0,4,2,0), can the request be granted immediately?

Sol.

Here, based on the snapshot of system,
Need = Max – Allocation
If need <= availability, then, available = available + allocation

T0: Available – 1520, Need – 0000
Need is less than Available, 0000 <= 1520 - True
We see that T0 finishes as true and put T0 in safe sequence.

T1: Available – 1520 + 0012 = 1532, Need – 750
Need is less than Available, 750 <= 1532 - True
We see that T1 finishes as true and put T1 in safe sequence.

T2: Available – 1532 + 1000 = 2532, Need – 1002
Need is less than Available, 1750 <= 2532 - True
We see that T2 finishes as true and put T2 in safe sequence.

T3: Available – 2532 + 1354 = 3886, Need – 0020
Need is less than Available, 0020 <= 3886 - True
We see that T3 finishes as true and put T3 in safe sequence.

T4: Available – 3886 + 0632 = 4518, Need – 0642
Need is less than Available, 0642 <= 4518 - True
We see that T4 finishes as true and put T4 in safe sequence.

a) Values of need:
   a. T0 = 0000
   b. T1 = 0750
   c. T2 = 1002
   d. T3 = 0020
   e. T4 = 0642

b) Yes, the system is in a safe state.
   With Available being 1520, we can start from process T0.
   Once process T0 runs, it releases its resources, which will allow other processes to run.

   The safe sequence will be T0, T1, T2, T3, T4.

c) The system receives a request for T1 arriving at 0420.

   We will check that, request (T1) <= need.
   The request for T1 is 0420 <= 0750 which means the condition request <= need is true.
   We also need to check that request(T1) <= available. Since, it is also true.

   So, this request can be granted by the system immediately, and the safe sequence remains the same. The system continues to be in a safe state.

Q3. No. 8.11.5

(a) Prove that the safety algorithm presented in Section Banker's algorithm requires an order of $m \times n^2$ operations.

Sol.

The resource-allocation-graph algorithm is not applicable to a resource-allocation system with multiple instances of each resource type. The deadlock-avoidance algorithm that we describe next is applicable to such a system but is less efficient than the resource-allocation graph scheme. This algorithm is commonly known as the banker's algorithm.

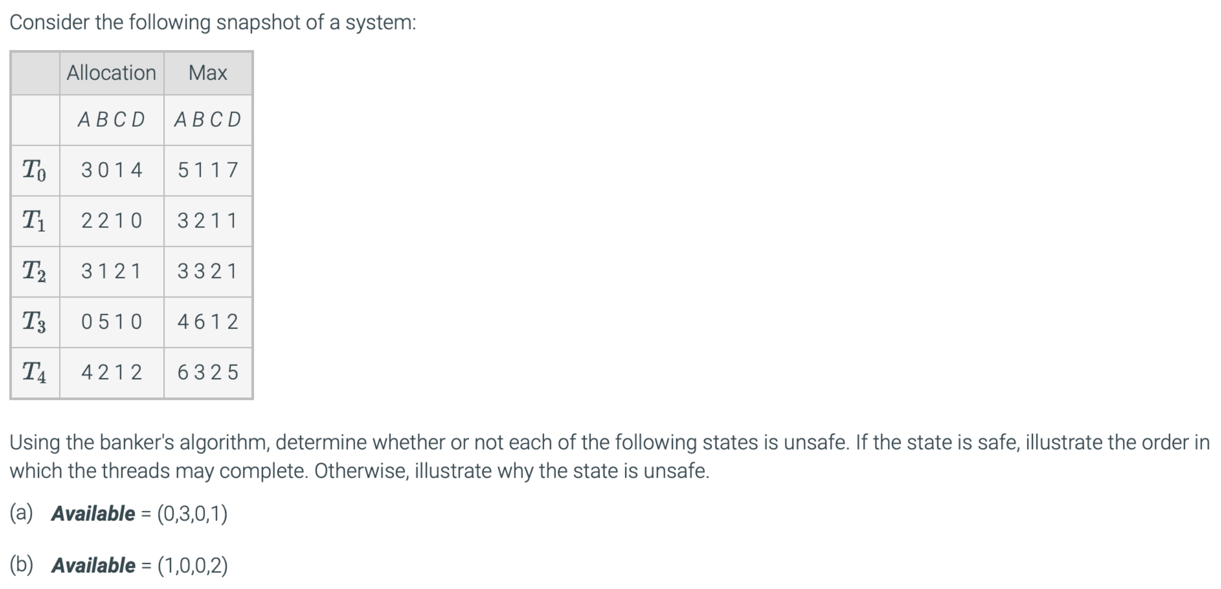
The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities, before deciding whether allocation should be allowed to continue.

Banker's Algorithm Code:

```
for(int I = 0; I < n; i++)
{
        // first find a thread that can finish
        for(int j = 0; j < n; j++) {

                if (!finish[j]) {
                        boolean temp = true;

                        for(int k = 0; k < m; k++){
                                if (need[j][k] > work[k])
                                temp = false;
                        }
                        if (temp) {
                        // if this thread can finish
                        finish[j] = true;

                        for (int x = 0; x < m; x++)
                                work[x] += work[j][x];
                        }
                }
        }
}
```

The nested outer loops, both of which loop through n times, provide the n^2 performance. Two sequential inner loops, loop m times within these outer loops. So, the total run-time of the algorithm will be $O(m \times n^2)$.

Q4. No. 8.11.9



Consider the following snapshot of a system:

	Allocation	Max
	A B C D	*A B C D*
T_0	3 0 1 4	5 1 1 7
T_1	2 2 1 0	3 2 1 1
T_2	3 1 2 1	3 3 2 1
T_3	0 5 1 0	4 6 1 2
T_4	4 2 1 2	6 3 2 5

Using the banker's algorithm, determine whether or not each of the following states is unsafe. If the state is safe, illustrate the order in which the threads may complete. Otherwise, illustrate why the state is unsafe.

(a) ***Available*** = (0,3,0,1)

(b) ***Available*** = (1,0,0,2)

Sol.

Here, based on the snapshot of system,

Need = Max – Allocation

	Allocation	**Max**	**Need = Max - Allocation**
Process	**ABCD**	**ABCD**	**ABCD**
T0	3014	5117	2103
T1	2210	3211	**1001**
T2	3121	3321	**0200**
T3	0510	4612	**4102**
T4	4212	6325	2113

a)

Here, it is required to check that need <= availability

Availability = 0301.

If need <= availability, then available = available + allocation.

Here, we start with process T2, as need for process T2 = 200 which is less than availability = 0301.

Now, Availability = 0301 + 3121 = 3422.

We again check whether need <= availability, so, need for process T1 - 1001 <= 3422 - Availability .

Now, Availability = 3422 + 2210 = 5632.

We again check whether need <= availability, so, need for process T3 - 4102 <= 5632 - Availability .

Now, Availability = 5632 + 0510 = 51142 (Here B, gets 11)

Now, the processes T0 and T4 cannot be finished because need > availability in some instances of resources ABCD.

So, the given state 0301 is not in safe state.

b)

Here, it is required to check that need <= availability

Availability = 1002.

If need <= availability, then available = available + allocation.

Here, we start with process T1, as need for process T1 = 1001 which is less than availability = 1002.

Now, Availability = 1002 + 2210 = 3212.

We again check whether need <= availability, so, need for process T2 - 0200 <= 3212 - Availability.

Now, Availability = 3212 + 3121 = 6333.

We again check whether need <= availability, so, need for process T3 - 4102 <= 6333 - Availability.

Now, Availability = 6333 + 0510 = 6843.

We again check whether need <= availability, so, need for process T0 - 2103 <= 6843 – Availability.

Now, Availability = 6843 + 3014 = 9847.

We again check whether need <= availability, so, need for process T4 - 2113<= 9847 - Availability.

Now, Availability = 9847 + 4212 = 13 10 5 9.

So, the given state 1002 is in safe state.

The safe sequence will be T1, T2, T3, T0, T4.

Q5. In an electronic funds transfer system, there are hundreds of identical processes that work as follows. Each process reads an input line specifying an amount of money, the account to be credited, and the account to be debited. Then it locks both accounts and transfers the money, releasing the locks when done. With many processes running in parallel, there is a very real danger that having locked account x it will be unable to lock y because y has been locked by a process now waiting for x. Devise a scheme that avoids deadlocks.
A bit more precise, as a programmer you are given the API that consists of two procedures: Lock(char[] account_name) and Release(char[] account_name). Your job is to write, in pseudo-code, the procedure Transfer (char[] account_from, char[] account_two, int amount). Do not release an account record until you have completed the transactions. (In other words, solutions that lock one account and then release it immediately if the other is locked, are incorrect.) You can use only the above API. (30 points)

Sol.

Deadlock is possible if two threads simultaneously invoke the transaction() function, transposing different accounts. That is, one thread might invoke

```
transaction(checking_account, savings_account, 25.0)
```

and another might invoke

```
transaction(savings_account, checking_account, 50.0)
```

```
void transaction(Account from, Account to, double amount)
{
        mutex lock1, lock2;
        lock1 = get_lock(from);
        lock2 = get_lock(to);

        acquire(lock1);
        acquire(lock2);

        withdraw(from, amount);
        deposit(to, amount);

        release(lock2);
        release(lock1);
}
```

Q6. Can a system be in a state that is neither deadlocked nor safe? If so, give an example. If not, prove that all states are either deadlocked or safe.

Sol.

Yes, a system could be in an unsafe state and not be deadlocked condition is possible, but it may lead to be deadlocked states. Being in an unsafe state implies that there is no guarantee that the system will never get in a deadlock.

For example, we have four resources as topes, plotters, scanners, and CD – ROMs as in text, three processes competing for them.

We may have the following situation.

Process	Allocation	Needs	Available
P0	2000	1020	0121
P1	1000	0131	
P2	0121	1010	

This is an example where the system is not deadlocked because many actions can still occur.

For example, A can still get 2 printers.

However, if each process asks for its remaining requirements, we have a deadlock.