Roll Num: Name:

CS3523: Operating Systems 2

Quiz3 – Spring 2022

Instructions: Please type your answers

1. Consider the following snapshot of a system:

Processes	**Allocation**				**Max**			
	A	**B**	**C**	**D**	**A**	**B**	**C**	**D**
P0	0	0	1	2	0	0	1	2
P1	1	0	0	0	1	7	5	0
P2	1	3	5	4	2	3	5	6
P3	0	6	3	2	0	6	5	2
P4	0	0	1	4	0	6	5	6

(a) You are given that Available = (1, 5, 2, 0). Using the banker's algorithm, show that the state is safe.

Processes	**Allocation**				**Max**			
	A	**B**	**C**	**D**	**A**	**B**	**C**	**D**
P0	0	0	1	2	0	0	1	2
P1	1	0	0	0	1	7	5	2
P2	1	3	5	4	2	3	5	6
P3	0	6	3	2	0	6	5	2
P4	0	0	1	4	0	6	5	6

(b) You are given that Available = (1, 5, 2, 0). Consider the case that P1 requests for the resources **(0,4,2,1)**. Can this be granted immediately?

(6 pts)

2. Explain how the safety algorithm discussed in the class requires an order of $m \times n^2$ operations.

(5 pts).

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.

(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.

(6 pts)

Please note that you must ensure that your solution is decentralized.

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?

(5 pts)

5. Consider the code below which is figure 8.7 of the book:

```
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);
}
```

The book showed that the invocation of this method by two threads can cause a deadlock. Suppose thread1 invokes the following:
transaction(checking account, savings account, 25.0)

While thread2 might invoke the following:
transaction(savings account, checking account, 50.0)

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

(5 pts)