Operating System 2: Exam 3

16th March 2022

Tanmay Garg CS20BTECH11063

Q1.

Sol.

(a)

- Need matrix will be (0000), (0750), (1002), (0020), (0642)
- P0 will run first as need is 0000
- Then work becomes (1532)
- P2 will run next and work becomes (2886)
- P1 will run next then P3 and then P4
- The system is in the safe state

(b)

- No this not cannot be granted as the new value of available will become undefined when using the procedure from Bankers Algorithm
- New available matrix value needs to be updated as (1520) - (0421) which is not defined

Q2.

Sol.

```
for (i = 0; i < n; i++)
{
        for(j = 0; j < n; j++)
        {
                if(finish[j] == false)
                {
                        tmp = true
                        for(k = 0; k < m; k++)
                        {
                                if(need[j][k] > work[k])
                                {
                                        tmp = false;
                                }
                        }
                        if(tmp == true)
                        {
                                finish[j] = true;
                                for(a = 0; a < m; a++)
                                {
                                        work[a] += work[j][a]
                                }
                        }
                }
        }
}
```

- The above is the pseudocode for the banker's algorithm
- The nested Outer loops go on for n times each hence they will take n^2 time
- There are some loops inside which go on for m times
- The total running time is of the order of $O(mn^2)$

Q3.

Sol.

(a)

- In order to implement banker's algorithm, the variables Max, Allocation and Need can be kept global and each variable should have a mutex lock associated with it to avoid multiple reads and writes at the same time
- Request will be granted only if the system passes the safety check
- So there can be multiple threads that might access the same global variables, so there needs to be some safety feature for each of the variables

(b)

- We can use multiple mutex locks for each of the variables discussed above
- These locks will keep the variables occupied by only one thread when needed
- These variables will be released once the safety algorithm can determine if the system is in a safe state

Q4.

Sol.

- The condition of the system is that we have a total of p processes
- Each process requires a maximum of m resources
- Total available resources are r
- In order to make the system deadlock free, there must be some conditions on the each of the variables to make it deadlock free
- Since all processes require resources 1 less than maximum resources
    - $total\ required = p(m-1)$
- This is maximum utilized, so we will need 1 more resource to avoid deadlock completely if the system is maxed out
    - $r \geq p(m-1)+1$
- Hence, the above is the condition to get a deadlock free system

Q5.

Sol.

```
transaction(Account from, Account to, int amount)
{
        mutex lock1, lock2;
        mutex l3;
        l3 = get_lock2(from, to);
        acquire(l3);
        lock1 = get_lock(from);
        lock2 = get_lock(to);
        acquire(lock1);
        acquire(lock2);
        withdraw(from, amount);
        deposit(to, amount);
        release(l3);
        release(lock2);
        release(lock1);

}
```

- This is our modified transaction function
- We have added an extra mutex lock l3 which will be acquired before acquiring lock1 and lock2
- This lock will be released when before releasing lock2 and lock1
- Get_lock2 function returns the lock associated with the 2 accounts taken into consideration, in some form which can be used by mutex to acquire that lock
- This get_lock2 function will allow safety for either or both the accounts on which transaction is happening
- For eg. If one thread calls the transaction function on account1 and account2, and other thread also calls the function on the same accounts
- Then get_lock2 will return something associated with both accounts, then l3 will try to acquire it and once acquired it can proceed with the transaction
- If there is any other thread that wants to do some transaction on any of these two accounts, then it will have to wait till our mutex lock3 releases this lock associated with the account
- The individual locks of the accounts are also there, which will be released once the transaction is completed
- This system allows for multiple transactions occurring on different accounts at the same time and the transactions in which some similar account is being considered will occur on the basis of as and when it acquires all the locks to continue with the transaction
- Some of other function internally used can be modified in order to use this type of procedure or a derived class can be made to make this algorithm work with all the required conditions