CS3523 - Operating Systems - 2
Quiz3 - Spring 2022
Vanga Aravind Shounik - CS20BTECH11055

1. Given Snapshot of the system:
   a)

Processes	Allocation	Max	Need	Available
	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	0 0 0 0	1 5 2 0
P1	1 0 0 0	1 7 5 0	0 7 5 0	1 5 3 2
P2	1 3 5 4	2 3 5 6	1 0 0 2	2 8 8 6
P3	0 6 3 2	0 6 5 2	0 0 2 0	2 14 11 8
P4	0 0 1 4	0 6 5 6	0 6 4 2	2 14 12 12
	2 9 10 12			3 14 12 12

   Here, we can see that using banker's algorithm, we can find a safe state for the process to execute in which is P0,P2,P3,P4,P1. Here, if the processes execute in this order, then after P0 executes, we get the available resources to (1,5,3,2) , Now, P2 executes and then we get the available resources to (2,8,8,6), Now, P3 can execute and get the available resources to (2,14,11,8), Now, P4 executes and gets the available resources to (2,14,12,12), Now, P1 executes and gets the available resources to (3,14,12,12) which is the total number of resources. So, we can say that the state is safe.

   b)
   Here, if P1 requests for (0,4,2,1) then we can't grant the resources immediately because there are not enough resources of D i.e, it asked for 1 resource but we have 0 resources left of D. So, it is not possible to grant the resources to P1 immediately.

2. Algorithm:
   Step1:
   Let Work and Finish be vectors of length m and n, respectively. Initialize Work = Available. Now for i = 0,1,...n-1, if Allocation_i != 0 , then Finish[i] = false, else Finish[i] = true.

   Step 2:
   Find an index such that Finish[i]==false and Request_i<=Work.If no such i exists, go to step 4. Here the maximum time taken is n.
   Step 3:
   Work = Work + Allocation
   Finish[i]=true
   Go to step2
   Step 4:
   If finish[i]==false for some i, 0<=i<n, then the system is in deadlocked state. Else it is a safe state.

   Here, we can see that we come to step 2 at most ‘n’ times. And each time we execute that statement, it takes time ‘n’. If it goes to step 3, then it takes time ‘m’ to execute the statement because it has to change ‘m’ values in the Work vector. So, the total time is in the order of O(m*n^2).

3. Here, if we wish to implement Banker’s algorithm for Deadlock Avoidance, then
   a. Here, the data structures required to store Max, Allocation, Need are vectors and they should be stored in the shared memory between threads so that each thread can access the vectors and they can change the data.
   b. Here, we can see that because the data structures are stored in the shared memory between the threads, we can say that each thread can access the vectors. Now, to prevent 2 threads accessing the data structures at the same time and changing the values which may lead to loss of data, we should make sure that the vectors are atomic. So, that only one operation can be done on them at a time.

4. Here, given there are ‘p’ processes each needing a maximum of ‘m’ resources. Total number of resources is ‘r’. Here, if k resources are allocated to each process, then the total number of resources allocated is p*(k) and we would need extra m-k resources to complete the process. Here, we can say that r should be

greater than or equal to p*k + m - k. Here, we took a case favorable to us. If we take the worst case then it should be when k = m-1 because we have to take into consideration that each process asks almost all the needed resources and if we can allocate them, then we can prevent deadlock. So, we can say that

$$r>=p*(m-1) + 1$$

Here, p*(m-1) is the number of resources allocated to all the processes and 1 is the number of resources we need to allocate to each process to run i.e, need.

5. Here, we can solve this using a deadlock prevention algorithm, here, we can prevent the deadlock by using the get_lock function in a lexicological order (or) the order in which the data is stored in its database. I.e, according to the id of the bank account

   Here, it is like

```
void transaction(Account from,Account to, double amount)
{
mutex lock1,lock2;
if(from.id>to.id)
{
     lock2=get_lock(to);
     lock1=get_lock(from);
}
else
{
     lock1=get_lock(from);
     lock2=get_lock(to);
}
acquire(lock1);
acquire(lock2);
withdraw(from, amount);
deposit(to, amount);
release(lock2);
release(lock1);
}
```

   Using this, we can make sure that the locks are taken in order and the deadlock can be prevented in the given case.