Operating Systems 2 (CS3523) Quiz 3

SURAJ TELUGU

CS20BTECH11050

1.(a). Sol

	Allocation				Max				Available				Need			
Process	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
T0	0	0	1	2	0	0	1	2	**1**	**5**	**2**	**0**	0	0	0	0
T1	1	0	0	0	1	7	5	0					1	7	5	0
T2	1	3	5	4	2	3	5	6					1	0	0	2
T3	0	6	3	2	0	6	5	2					0	0	2	0
T4	0	0	1	4	0	6	5	6					0	6	4	2

Applying Bankers' Safety Algorithm:

Let Work be and Finish be vectors of length 4 and 5, respectively

Initialisation:

Work = Available Finish[i] = false; for i = 1, 2, 3, 4, 5

First T0 starts execution since Need[0] <= Work. T0 executes without any additional resources. after execution:

Work = Work + Allocation[0] = (1, 5, 3, 2)

Next T2 starts execution since Need[2] <= Work. T2 executes with 1 instance of A and 2 instances of D from Work (Available). after execution:

Work = Work + Allocation[2] = (2, 8, 8, 6)

Next T3 starts execution since Need[3] <= Work. T3 executes with 2 instances of C from Work (Available). after execution:

Work = Work + Allocation[3] = (2, 14, 11, 8)

Next T4 starts execution since Need[4] <= Work. T4 executes with 6 instances of B, 4 instances of C, 2 instances of B from Work (Available). after execution:

Work = Work + Allocation[4] = (2, 14, 12, 12)

Next T1 starts execution since Need[1] <= Work. T1 executes with 1 instance of A, 7 instances of B, 5 instances of C from Work (Available). after execution:

Work = Work + Allocation[5] = (3, 14, 12, 12)

The Safe Sequence can be <T0, T2, T3, T4, T1>. Since a safe sequence exists the given system is therefore, in a safe state.

1.(b). Sol

	Allocation				Max				Available				Need			
Process	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
T0	0	0	1	2	0	0	1	2	**1**	**5**	**2**	**0**	0	0	0	0
T1	1	0	0	0	1	7	5	0					1	7	5	2
T2	1	3	5	4	2	3	5	6					1	0	0	2
T3	0	6	3	2	0	6	5	2					0	0	2	0
T4	0	0	1	4	0	6	5	6					0	6	4	2

Applying Bankers' Resource – Request Algorithm:

The request can be granted immediately only if request resource is less than Need[1] and also less than Available

Request[1] = (0, 4, 2, 1) Need[1] = (1, 7, 5, 2) Available = (1, 5, 2, 0)

Request[1] < Need[1] but Request[1] > Available the process 1 has to wait since its resources are not available

The system is in unsafe state it is better not to grant the request of P1 immediately. The system might get deadlock if the request gets granted.

2. Sol

In the safety algorithm of bankers algorithm for implementation of step 2 we need 3 for loops which are nested. First loop runs on i = 0 to n and starts to search for required i it takes O(n) time. Inside First loop, second loop checks which i has Finished[i] == false it take O(n) time. The third loop lies inside second loop which checks Need[i] ≤ Work condition it runs for i=0 to m so it takes O(m) times. Since the loops are nested overall time taken for safety algorithm is $O(m \times n^2)$ (Big – Oh (Number of different resources x $(\text{Number of threads})^2$).

3. Sol

3.(a). Sol

The data structures Max, Allocation, Need should be defined globally and can be initialised with dynamic allocation as n x m 2d arrays. Once we get the values of number of resources (m) and number of processes (n).

If we are given Max and Allocation we can Need using

Need [i,j] = Max[i,j] – Allocation [i,j]

We can store them and lock their access with mutex lock so that no two threads can simultaneously alter the Allocation matrix after getting its request granted. If they are altered simultaneously, it may give wrong results.

3.(b). Sol

The process should not get access to Allocation matrix or Need matrix simultaneously. If two process have requested resources simultaneously they both together may alter the data in matrices giving false data and results.

Therefore they should be locked such that only one process request can be accepted and that process can modify the Need and Allocation matrix.

To Ensure correct working after every alteration of data we need to check whether Total Resources = Available + ((∑Allocation[i][j]) for i = 0,1, 2, 3, 4 is constant or not. If not the algorithm is not working properly.

4. Sol

Given the system,

Number of Processes = p

Maximum resource needed by each process = m

Total Number of available resources = r

For the system to be deadlock free it has to be in safe state after each process is executed.

Consider extreme case where Allocation[i] = m - 1 every process is allocated with m - 1 resources. So Need[i] = 1 every process needs 1 more resource to execute.

If there are no resources left after allocation then every process waits for the last resource and no process executes ending up in a deadlock state. To avoid deadlock and satisfy Need[i]. Available = 1 after allocation.

r (Available) ≥ p (Number of Processes) x (m – 1) (Maximum – 1)

5. Sol

```
int Func(lock)
{
        if(lock == lock1) return 1;
        else if(lock == lock2) return 2;
}

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