Ex No. 9 Deadlock detection

Dr S.Rajarjan
SOC/SASTRA

Steps to be followed

Step1 : Take input for Max, Allocation and Available
Step 2: Find a process which can be completed with available resources
Step 3: If no process can be found then go to step 5
Step 3: Mark that process as completed and add its allocated resources with available resources
Step 4: Go to step 2
Step 5: If all the processes are completed then show the result as SAFE, otherwise show the result as UNSAFE

Sample data

	Current Allocation				Maximum Need			
	A	B	C	D	A	B	C	D
P0	2	0	2	1	9	5	5	5
P1	0	1	1	1	2	2	3	3
P2	4	1	0	2	7	5	4	4
P3	1	0	0	1	3	3	3	2
P4	1	1	0	0	5	2	2	1
P5	1	0	1	1	4	4	4	4

Available Resources

6	3	5	4

```c
#include <stdio.h>
int main()
{

    int n, m, i, j, k, avail[10],alloc[10][10],max[10][10],need[10][10];
    printf("Enter number of processes: ");
    scanf("%d",&n);
    printf("Enter number of resources: ");
    scanf("%d",&m);
    printf("Enter the number of available resources:\n");
    for(i=0;i<m;i++)
    {
        printf("Resource%d: ",i);
        scanf("%d",&avail[i]);
    }
    printf("\nEnter maximum resource demand of each process:\n");
    for(i=0;i<n;i++)
    {
        printf("Process %d :\n",i);
        for(j=0;j<m;j++)
        {
            scanf("%d",&max[i][j]);
        }
    }
     printf("\nEnter already allocated resources of each process:\n");
      for(i=0;i<n;i++)
      {
        printf("Process %d:",i);
        for(j=0;j<m;j++)
        {
            scanf("%d",&alloc[i][j]);
        }
    }

    int f[n], ans[n], ind = 0;
    for (k = 0; k < n; k++) {
        f[k] = 0; // to be used to trace which proceeses have completed
    }
    //Calculate the current need of all the processes
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++)
            need[i][j] = max[i][j] - alloc[i][j];
    }
    int y = 0;
    for (k = 0; k < n; k++) {
        for (i = 0; i < n; i++) {
            if (f[i] == 0) {
                int flag = 0;
                for (j = 0; j < m; j++) {
                    if (need[i][j] > avail[j])//Need exceeds availability
```

```
                    {
                        flag = 1; // ith process can not be completed
                         break;
                    }
                }
                if (flag == 0)
                {
                    ans[ind++] = i;// Constructing safe sequence
                    for (y = 0; y < m; y++)
                        avail[y] += alloc[i][y]; // Reclaim the resource from completed p
                    f[i] = 1; // Update that ith process is completed
                }
            }
        }
    }
    int flag = 1;
    for(int i=0;i<n;i++)
    {
        if(f[i]==0) // If a process could not be completed
        {
            flag=0;
            printf("The following system is not safe");
            break;
        }
    }
    if(flag==1) //All the processes can be completed
    {
      printf("Following is the SAFE Sequence\n");
      for (i = 0; i < n - 1; i++)
        printf(" P%d ->", ans[i]);
    printf(" P%d", ans[n - 1]);
    }
    return (0);
}
```

```
Enter number of processes: 6
Enter number of resources: 4
Enter the number of available resources:
Resource0: 6
Resource1: 3
Resource2: 5
Resource3: 4

Enter maximum resource demand of each process:
Process 0 :
9 5 5 5
Process 1 :
2 2 3 3
Process 2 :
7 5 4 4
Process 3 :
3 3 3 2
Process 4 :
5 2 2 1
Process 5 :
4 4 4 4

Enter already allocated resources of each process:
Process 0:2 0 2 1
Process 1:0 1 1 1
Process 2:4 1 0 2
Process 3:1 0 0 1
Process 4:1 1 0 0
Process 5:1 0 1 1
Following is the SAFE Sequence
 P1 -> P2 -> P3 -> P4 -> P5 -> P0
--------------------------------
```