Operating System Report

Student Name : Anshal Singh

Student ID : 11802648

Section :K18FG

Roll No : 47

Email Address : Anshal.54321@gmail.com

Github Link : https://github.com/Anshal55/OS

Question:

Ques. 22. Consider following and Generate a solution to find whether the system is in safe state or not?

Available				Processes	Allocation				Max			
A	B	C	D		A	B	C	D	A	B	C	D
1	5	2	0	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

Description:

In the problem given I have to find whether the system is in a safe state or not.

What is a safe state?

When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state. System is in safe state if there exists a safe sequence of all processes.

What is an unsafe state?

If there is a chance for the system to go into a deadlock then it is an unsafe state.

Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

To find whether the system is in a safe state or not we can use safety algorithm where we check whether the resources needed are more than available.

Algorithm:

Safety Algorithm:

1) Let Work and Finish be vectors of length ‘m’ and ‘n’ respectively.
Initialize: Work = Available
Finish[i] = false; for i=1, 2, 3, 4….n

2) Find an i such that both
a) Finish[i] = false
b) $Need_i$ <= Work
if no such i exists goto step (4)
3) Work = Work + Allocation[i]
Finish[i] = true
goto step (2)

4) if Finish [i] = true for all i
then the system is in a safe state

Resource-Request Algorithm:
1) If $Request_i$ <= $Need_i$
Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.
2) If $Request_i$ <= Available
Goto step (3); otherwise, P_i must wait, since the resources are not available.
3) Have the system pretend to have allocated the requested resources to process Pi by modifying the state as
follows:
Available = Available – Requesti
$Allocation_i$ = $Allocation_i$ + $Request_i$
$Need_i$ = $Need_i$– $Request_i$

Code Snippets:

```c
//A program to determine whether the system is in a safe state or not.
#include<stdio.h>
int i,j,t; //Initialization
int count =0;
bool bol=false;
int main()
{
    printf("Enter how many resources: \n");
    int a;
    scanf("%d",&a);
    int res[a];
    printf("Enter the total weight of all resources:\n");
    for(i=0;i<a;i++)
    {
        printf("%c is  ",65+i);
        scanf("%d",&res[i]);
    }
    printf("\nEnter how many processes:\n ");
    int b;
    scanf("%d",&b);
    int c[b][a],max[b][a],need[b][a],avali[a]={0};
    m:for(i=0;i<b;i++)
    for(j=0;j<a;j++)
    {
        printf("\nP[%d] allocation of %c:",i,(char)65+j);
        scanf("%d",&c[i][j]);
        printf("\nP[%d] maximum Required resource for %c:",i,(char)65+j);
        scanf("%d",&max[i][j]);
        need[i][j]=max[i][j]-c[i][j];
    }
    for(j=0;j<a;j++)
    {
    for(i=0;i<b;i++)
    {
        if(avali[j]>res[j])
        {
            printf("Error correct the allocation.\n");
            goto m;
```

```
                printf("Error correct the allocation.\n");
                goto m;
            }
            else
            avali[j]=avali[j]+c[i][j];
        }
        avali[j]=res[j]-avali[j];
        }
        i=0;
        while(count<b&&t<b)
        {
            bol=true;
            for(j=0;j<a;j++)
            {
                if(need[i][j]>avali[j])
                {
                    t++;
                    bol=false;
                    break;
                }
            }
            if(bol)
            {
                count++;
                for(j=0;j<a;j++)
                {
                avali[j]+=c[i][j];
                need[i][j]=res[j]+2000;
                }
            }
            if(i==b-1)
            i=0;
            else
            i++;
            }
        if(count==b)
        {
            printf("\nSafe.");
```

```
72      if(count==b)
73      {
74          printf("\nSafe.");
75      }
76      else
77      {
78          printf("\nUnsafe");
79          printf("\nThe safe state is:P0->P2->P3->P4->P1");
80      }
81  }
```