**LOVELY PROFESSIONAL UNIVERSITY**

**(Report)**

**Course Code:** CSE 316 **Course Title:** Operating System

**Student Name:** Bhavesh Gujjula

**Student ID:** 11801452

**Section:** K18NZ

**Email Address**: [gujjulabhavesh@gmail.com](mailto:gujjulabhavesh@gmail.com)

**GitHub link:**

**Code:**

**Question20:** Consider that a system has P resources of the same type. These resources are shared by Q processes from time to time. All processes request and release the resources one at a time. Generate a solution to demonstrate that, the system is in a safe state when the following conditions are satisfied.

Conditions:

1. The maximum resource need for each process is between 1 and P.

2. Summation of all maximum needs is less than P+Q

#include <stdio.h>

int main()

{

int n, m, i, j, k,alloc[15][15],max[15][15],avail[15];

printf("Enter Processes : ");

scanf("%d", &n);

printf("Enter Resources : ");

scanf("%d", &m);

printf("\Input the values of Allocation Matrix for the processes : ");

for(i = 0; i < n; i++)

{

printf("\nFor process %d : \n",i + 1);

for(j = 0; j < m; j++)

{

scanf("%d", &alloc[i][j]);

}

}

printf("\n\nInput the values of Maximum Matrix for the processes : \n");

for(i = 0; i < n; i++)

{

printf("\nFor process %d : \n", i + 1);

for(j = 0; j < m; j++)

{

scanf("%d", &max[i][j]);

}

}

printf("\n\n Give the Available Resources : \n");

for(i = 0; i < m; i++)

{

scanf("%d", &avail[i]);

}

int f[n], ans[n], ind = 0;

for (k = 0; k < n; k++)

{

f[k] = 0;

}

int need[n][m];

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 < 5; 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])

{

flag = 1;

break;

}

}

if (flag == 0)

{

ans[ind++] = i;

for (y = 0; y < m; y++)

avail[y] += alloc[i][y];

f[i] = 1;

}

}

}

}

int temp=1;

for(i=0;i<n;i++)

{

if(f[i]==0)

{

temp=0;

break;

}

}

if(temp==1)

{

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]);

}

else

{

printf("The given sequence is in unsafe state");

}

}

**Description:**

Proof:  
Suppose

N = Sum of all Need(i),

A = Sum of all Allocation(i),

M = Sum of all Max(i).

Use contradiction to prove.   
Assume this system is not deadlocked free. If there exists a deadlock state, then A = p, as a result, there is only one kind of resource and resources will be requested and released only one at a time. From the condition b, N + A = M < p + q. So, we tend to get N + p < p + q. So, we get N < q. It shows that a minimum of one process i that Need(i) = 0. From the condition, Pi can unharness a minimum of one resource. So, there are q-1 processes sharing p resources currently, condition a and b still hold. Proceed with the argument, no process will wait permanently, so there's no deadlock.

Also Using banker's algorithm you'll see that one process can acquire all required resources and when completion of work it will release resources hence no deadlock will occur.

**A close up of text on a whiteboard

Description automatically generatedAlgorithm:**

**Worst Complexity Case:**

Consider the worst-case- all processes acquire most resources but still not capable to complete. So, the resources available must be 1 less than the maximum need, for each of the processes (this ensures none of them will finish).  
We are given most need is always less than p + q. As per our condition for deadlock, resources available must be 1 less than the maximum need for each of 'q' processes --> the resources out there available must be less than p + q - p = q.  
But 'p' is that the available number of resources and hence no deadlock can occur.

**Conditions are given within the problem:**

The following conditions should be satisfied

a. The maximum need for each process is between one resource and p resources.

If the system is assumed to not be deadlock-free and there exists a deadlock state, then A=M because there is only one resource that might be requested/released one at a time.

b. The sum of all maximum needs is less than p+ q.

In this condition, M < p+q = N+A, which is that a similar as N+p< p+q so N < q. From the condition, this process n release a minimum of one resource, so there are q-1 processes sharing q resources at this time currently, and both conditions a and b still hold true. No processes can wait permanently thus there’s no deadlock.

**Example Problem:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Process** | **Allocation** | **Maximum** | **Available** | **Need** |
|  | A B C | A B C | A B C | A B C |
| P0 | 0 1 0 | 7 5 3 | 3 3 2 | 7 4 3 |
| P1 | 2 0 0 | 3 2 2 |  | 1 2 2 |
| P2 | 3 0 2 | 9 0 2 |  | 6 0 0 |
| P3 | 2 1 1 | 2 2 2 |  | 0 1 1 |
| P4 | 0 0 2 | 4 3 3 |  | 4 3 1 |
|  | **<7 2 5>** |  |  |  |

Available = [(10-7), (5-2), (7-5)] = < 3 3 2>

Need= Maximum – Allocation

Available = Available + Allocated

Safe Sequence<P1, P3, P4, P0, P2>

**The answer obtained from the code:**

A screenshot of a computer screen

Description automatically generated

The algorithm used to show the safe state is Banker’s Algorithm and Safe StateAlgorithm.

**Have you made a minimum of 5 revisions of the solution on GitHub?**

**Ans:** Yes

**GitHub Link:**