**ST. XAVIER’S COLLEGE**

**(Affiliated to Tribhuvan University)**

**Maitighar, Kathmandu**

****

**OPERATING SYSTEM LAB REPORT #11**

**SUBMITTED BY:**

Pradeep Dahal

017BSCIT029

2nd year/ 4th sem

|  |  |
| --- | --- |
|  | Signature |
| Mr. Rabin Maharjan  (Lecturer) |  |
| Department of Computer Science | |

**SUBMITTED TO:**

# Banker’s Algorithm

The banker’s algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an “s-state” check to test for possible activities, before deciding whether allocation should be allowed to continue.

Following Data structures are used to implement the Banker’s Algorithm:

Let ‘n’ be the number of processes in the system and ‘m’ be the number of resources types.

**Available :**

* It is a 1-d array of size ‘m’ indicating the number of available resources of each type.
* Available[ j ] = k means there are ‘k’ instances of resource type Rj

**Max :**

* It is a 2-d array of size ‘n\*m’ that defines the maximum demand of each process in a system.
* Max[ i, j ] = k means process Pi may request at most ‘k’ instances of resource type Rj.

**Allocation :**

* It is a 2-d array of size ‘n\*m’ that defines the number of resources of each type currently allocated to each process.
* Allocation[ i, j ] = k means process Pi is currently allocated ‘k’ instances of resource type Rj

**Need :**

* It is a 2-d array of size ‘n\*m’ that indicates the remaining resource need of each process.
* Need [ i,   j ] = k means process Pi currently need ‘k’ instances of resource type Rj

for its execution.

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

Allocationi specifies the resources currently allocated to process Pi and Needi specifies the additional resources that process Pi may still request to complete its task.

## SOURCE CODE:

// Banker's Algorithm

#include <stdio.h>

int main()

{

// P0, P1, P2, P3, P4 are the Process names here

int n, m, i, j, k,alloc[20][20],max[20][20],avail[20];

printf("\n Enter number of processes: ");

scanf("%d",&n);

// n = 5; // Number of processes

// m = 3; // Number of resources

printf("\n Enter number of resources: ");

scanf("%d",&m);

fflush(stdin);

printf("\n Enter the allocation matrix: \n ");

for(int a=0;a<n-1;a++){

for(int b=0;b<m;b++){

scanf("%d ", &alloc[a][b]);

}

}

/\* int alloc[5][3] = { { 0, 1, 0 }, // P0 // Allocation Matrix

{ 2, 0, 0 }, // P1

{ 3, 0, 2 }, // P2

{ 2, 1, 1 }, // P3

{ 0, 0, 2 } }; // P4

\*/ fflush(stdin);

printf("\n Enter the maximum resources allocated matrix:\n");

for(int a=0;a<n-1;a++){

for(int b=0;b<m;b++){

scanf("%d ", &max[a][b]);

}

}

/\*

int max[5][3] = { { 7, 5, 3 }, // P0 // MAX Matrix

{ 3, 2, 2 }, // P1

{ 9, 0, 2 }, // P2

{ 2, 2, 2 }, // P3

{ 4, 3, 3 } }; // P4

\*/

fflush(stdin);

printf("\n Enter the availabe resources: ");

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

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

}

fflush(stdin);

// int avail[3] = { 3, 3, 2 }; // Available Resources

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;

}

}

}

}

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);

}

