**A Mini Project Report**

**On**

**Banker's Algorithm**

**Submitted By :**

**(B-33)YASHRAJ PATIL**

**(B-25)CHINMAY PARADKAR**

**(B-39)SATISH SANGLE**

**Submitted in partial fulfillment of requirement of Operating Systems**

**UNDER THE GUIDANCE OF**

**Dr. Amol P. Pande**



**Department of Computer Engineering**

**DATTA MEGHE COLLEGE OF ENGINEERING**

**Sector 3,Airoli,Navi Mumbai-400708**

1. **Problem Statement**

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

1. **Objective**

**Deadlock Summary**

• Deadlock: situation in which a set of threads/processes cannot proceed because each requires resources held by another member of the set

• Detection and recovery: recognize deadlock after it has occurred and break it.

• Avoidance: don't allocate a resource if it would introduce a cycle.

• Prevention: design resource allocation strategies that guarantee that one of the necessary conditions never holds

Bankers Algorith is a Deadlock Prevention Type

**Use Of Bankers Algorithm in real life**

Bankers Algorithm is used in banking system to check whether loan can be sanctioned to a person or not. Suppose there are n number of account holders in a bank and the total sum of their money is S. If a person applies for a loan then the bank first subtracts the loan amount from the total money that bank has and if the remaining amount is greater than S then only the loan is sanctioned. It is done because if all the account holders comes to withdraw their money then the bank can easily do it.

In other words, the bank would never allocate its money in such a way that it can no longer satisfy the needs of all its customers. The bank would try to be in safe state always.

**Other types of algorithms to prevent deadlock**

* **Safety Algorithm**

The algorithm for finding out whether or not a system is in a safe state

* **Resource-Request Algorithm**

**This algorithm decides whether request of the resources can be safely granted so as to system will remain in a safe state .**

1. **Solution**

Algorithm :

Step 1 :Let W be a integer array of length m ,initialization of array a Let F be a Boolean array of length n ,initializrd to false .

Step 2 :F[i]==False

N[i]<=W

Step 3: W=W+C[i];

F[i]= true;

go to step 2

Step 4 :If F[i]== true for all i then the system is in a safe state

Example :

**Consider :**

|  |  |  |  |
| --- | --- | --- | --- |
| Process | Allocation | Max | Availaible |
|  | A B C | A B C | A B C |
| P0 | 0 1 0 | 7 5 3 | 3 3 2 |
| P1 | 2 0 0 | 3 2 2 |  |
| P2 | 3 0 2 | 9 0 2 |  |
| P3 | 2 1 1 | 2 2 2 |  |
| P4 | 0 0 2 | 4 3 3 |  |

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

|  |  |  |  |
| --- | --- | --- | --- |
| PROCESS |  | NEED |  |
|  | A | B | C |
| P0 | 7 | 4 | 3 |
| P1 | 1 | 2 | 2 |
| P2 | 6 | 0 | 0 |
| P3 | 0 | 1 | 1 |
| P4 | 4 | 3 | 1 |

Acoording to algorithm

Need<=Work

For P0 7 4 3 <= 3 3 2 False

For P1 1 2 2 <= 3 3 2 True

Work +allocation 3 3 2+2 0 0=5 3 2

New Work = 5 3 2

For P2 6 0 0<= 5 3 2 False

For P3 0 1 1<= 5 3 2 True

Work +Allocation 5 3 2+2 1 1=7 4 3

New Work = 7 4 3

For P4 4 3 1 <= 7 4 3 True

Work + allocation 7 4 3+0 0 2=7 4 5

New Work 7 4 5

For P0 7 4 3<= 7 4 5 True

Work + allocation 7 4 5+0 1 0=7 5 5

New Work 7 5 5

For P2 600<=755 True

Work +allocation 7 5 5+3 0 2=10 5 7

Therefore the correct sequence to execute or distribute the processes in the correct order so no deadlock occurs

The order for above Example is <P1,P3,P4,P0,P2>

1. Features used

**Available :**

* A[m] : Array A of size m shows the number of available resources

**Max :**

* M[n][m] : Two dimensional array M shows maximum requirement of the resource by each process

**Allocation :**

* C[n][m] : Two dimensional array C shows current allocation status of resources of each process

**Need :**

* N[m][n] : Two dimensional array N shows the remaining possible need of each process

1. **Platform Used**

Knowledge Source : Books and Online

Program type : C programming

Execution Source : Repl.it

1. **Source Code**

#include <stdio.h>

int main()

{

int n, m, i, j, k;

printf("Enter Number of Processes\n");

scanf("%d",&n);

printf("Enter Number of Resources\n");

scanf("%d",&m);

int alloc[n][m];

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

{

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

{

printf("Enter Resources to be allocated of type %d for P%d : ",j,i);

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

}

}

int max[n][m];

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

{

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

{

printf("Enter Max Resources of type %d for P%d : ",j,i);

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

}

}

int avail[m];

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

{

printf("Enter Available Resources of type %d : ",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 < 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])

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]);

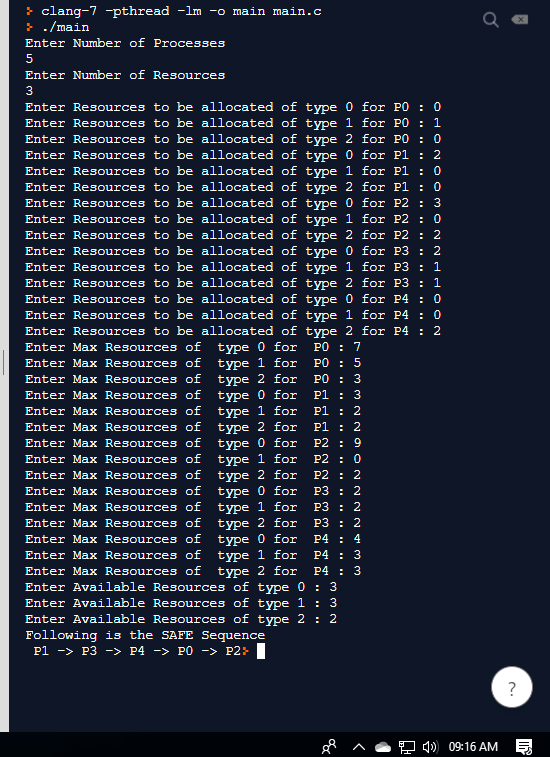
return (0);

}

**7.Input**

|  |  |  |  |
| --- | --- | --- | --- |
| Process | Allocation | Max | Availaible |
|  | A B C | A B C | A B C |
| P0 | 0 1 0 | 7 5 3 | 3 3 2 |
| P1 | 2 0 0 | 3 2 2 |  |
| P2 | 3 0 2 | 9 0 2 |  |
| P3 | 2 1 1 | 2 2 2 |  |
| P4 | 0 0 2 | 4 3 3 |  |

**8. OUTPUT:**

****

**Hece It is one of the succesfull method of Preventing a Deadlock .**