

**Lab no:**  **Date: 2079/**

**Title: Write a program to detect a deadlock using banker’s algorithm for multiple**

**resources of each type. Also if there is no deadlock display the safe sequence.**

**Banker’s Algorithm:**

The Banker algorithm is deadlock detection algorithm that tests for safety by simulating the allocation of predetermined maximum possible amounts of all resources, and then make an “safe-state” check to test for possible deadlock conditions for all other pending activities, before deciding whether allocation should be allowed to continue.

**Algorithm:**

### Banker’s algorithm comprises of two algorithms:

1. Safety algorithm
2. Resource request algorithm

**Safety algorithm:**

Step 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

Step 2: Find an i such that both

a) Finish[i] = false

b) Needi <= Work

if no such i exists goto step (4)

Step 3: Work = Work + Allocation[i]

Finish[i] = true

goto step (2)

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

**Resource request algorithm**

Step 1: If Requesti <= Needi

Goto step (2) ; otherwise, raise an error condition, since the process has

exceeded its maximum claim.

Step 2: If Requesti <= Available

Goto step (3); otherwise, Pi must wait, since the resources are not available.

Step 3: Have the system pretend to have allocated the requested resources to process

Pi by modifying the state as follows:

Available = Available – Requesti

Allocationi = Allocationi + Requesti

Needi = Needi– Requesti

Programming Language: C++

IDE: Dev-C++

**Code:**

|  |
| --- |
| //bankers algorithm  #include <iostream>  using namespace std;  int main()  {    int n, m, i, j, k, alloc[20][20],max[20][20],avail[20];  cout << " Enter the number of processes\n";  cin >> n;  cout << "Enter the number of resources\n";  cin >> m;  cout << "Enter the allocation matrix\n";  for (i=0;i<n;i++)  {  for(j=0;j<m;j++)  {  cin>>alloc[i][j];  }  }  cout << "Enter the max matrix\n";  for(i=0;i<n;i++)  {  for(j=0;j<m;j++)  {  cin>>max[i][j];  }  }  cout << "Enter the available resources\n";  for(i=0;i<m;i++)  {  cin>>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 flag = 1;  // check if sequence is safe or not  for(int i = 0;i<n;i++)  {  if(f[i]==0)  {  flag = 0;  cout << "The given Sequence is not safe";  break;  }  }  if(flag==1)  {  cout << "The given sequence is safe" << endl;  for (i = 0; i < n - 1; i++)  cout << " P" << ans[i] << " --->";  cout << " P" << ans[n - 1] <<endl;  }  return 0;  } |

**Output:**

****

**Fig:Bankers Algorithm**