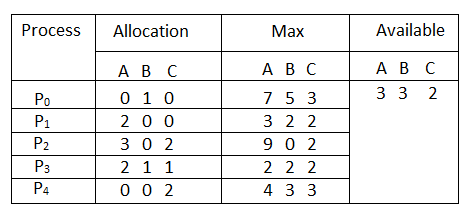
Experiment No: 3

Experiment Name:Implementation of Banker's Algorithm

Objectives: Learn about Bankers algorithm. Implement Bankers algorithm by using c program. And testing the program different input and find output .

Bankers algorithm:

The Banker's algorithm, sometimes referred to as the avoidance algorithm, is a resource allocation and deadlock avoidance algorithm developed by Edsger Dijkstra that tests for safety by simulating the allocation of predetermined maximum possible amounts of all resources, and then makes an "s-state" check to test for possible deadlock conditions for all other pending activities, before deciding whether allocation should be allowed to continue.

Considering a system with five processes P0 through P4 and three resources types A, B, C. Resource type A has 10 instances, B has 5 instances and type C has 7 instances. Suppose at time t0 following snapshot of the system has been taken:  


We must determine whether the new system state is safe. To do so, we need to execute Safety algorithm on the above given allocation chart.

Solve:

Executing safety algorithm shows that sequence < P1, P3, P4, P0, P2> satisfies safety requirement

Source Code :

// C++ program to illustrate Banker's Algorithm

#include<iostream>

using namespace std;

// Number of processes

const int P = 5;

//process sequence

const string safe[]={"P0","P1","P2","P3","P4"};

// Number of resources

const int R = 3;

// Function to find the need of each process

void calculateNeed(int need[P][R], int maxm[P][R],

int allot[P][R])

{

// Calculating Need of each P

for (int i = 0 ; i < P ; i++)

for (int j = 0 ; j < R ; j++)

// Need of instance = maxm instance -

// allocated instance

need[i][j] = maxm[i][j] - allot[i][j];

}

// Function to find the system is in safe state or not

bool isSafe(int processes[], int avail[], int maxm[][R],

int allot[][R])

{

int need[P][R];

// Function to calculate need matrix

calculateNeed(need, maxm, allot);

// Mark all processes as infinish

bool finish[P] = {0};

// To store safe sequence

int safeSeq[P];

// Make a copy of available resources

int work[R];

for (int i = 0; i < R ; i++)

work[i] = avail[i];

// While all processes are not finished

// or system is not in safe state.

int count = 0;

while (count < P)

{

// Find a process which is not finish and

// whose needs can be satisfied with current

// work[] resources.

bool found = false;

for (int p = 0; p < P; p++)

{

// First check if a process is finished,

// if no, go for next condition

if (finish[p] == 0)

{

// Check if for all resources of

// current P need is less

// than work

int j;

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

if (need[p][j] > work[j])

break;

// If all needs of p were satisfied.

if (j == R)

{

// Add the allocated resources of

// current P to the available/work

// resources i.e.free the resources

for (int k = 0 ; k < R ; k++)

work[k] += allot[p][k];

// Add this process to safe sequence.

safeSeq[count++] = p;

// Mark this p as finished

finish[p] = 1;

found = true;

}

}

}

// If we could not find a next process in safe

// sequence.

if (found == false)

{

cout << "System is not in safe state";

return false;

}

}

// If system is in safe state then

// safe sequence will be as below

cout << "System is in safe state.\nSafe"

" sequence is: \n\n";

for (int i = 0; i < P; i++)

{

cout << safeSeq[i];

if(i<P-1)

{

cout<<" --> ";

}

}

cout<<"\n\n\n";

return true;

}

// Driver code

int main()

{

int processes[] = {0, 1, 2, 3, 4};

// Available instances of resources

int avail[] = {3, 3, 2};

// Maximum R that can be allocated

// to processes

int maxm[][R] = {{7, 5, 3},

{3, 2, 2},

{9, 0, 2},

{2, 2, 2},

{4, 3, 3}

};

// Resources allocated to processes

int allot[][R] = {{0, 1, 0},

{2, 0, 0},

{3, 0, 2},

{2, 1, 1},

{0, 0, 2}

};

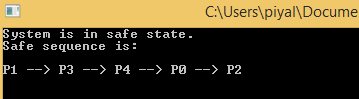
// Check system is in safe state or not

isSafe(processes, avail, maxm, allot);

return 0;

}

Output:



Discussion: In Lab experiment, We implement Banker's Algorithm according to our Class lecture.And in this experiment ,we get proper output.