**Student Name:** Shubham kalihari

**Student ID:** 11710408

**Email Address:** shubhamkalihari09@gmail.com

**GitHub Link:** https://github.com/starboysk/ospro11710408.git

**Code:**

#include<iostream>

using namespace std;

const int np = 5;

const int nr = 4;

static bool finish[np];

int i;

int j;

int ss[5];

int count=0;

int count2=0;

static int avail[nr] = {1,5,2,0};

static int work[nr];

static int allo[np][nr] = {{0,0,1,2},

{1,0,0,0},

{1,3,5,4},

{0,6,3,2},

{0,0,1,4}};

static int mx[np][nr] = {{0,0,1,2},

{1,7,5,0},

{2,3,5,6},

{0,6,5,2},

{0,6,5,6}};

static int need[np][nr];

void calneed(int all[np][nr],int mm[np][nr]){ //function to calculate need

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

for(j = 0; j<nr; j++){

need[i][j]=mm[i][j]-all[i][j];

}

}

}

bool check(int a,int temp[nr]){ //function to check weather need<work

int count;

for(int k=0;k<nr;k++){

if(need[a][k]<=temp[k]){

count=count+1;

}

}

if(count==nr){

return true;

}

else{

return false;

}

}

void add(int t){ //function to add alocation to work

for(j=0;j<nr;j++){

work[j]=allo[t][j]+work[j];

}

}

int main(){

//int count3=0;

int temp; // Step 1

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

work[i] = avail[i];

}

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

finish[i] = false;

} // step 1 finished

calneed(allo,mx);

while(count2<np) //loop will run until all the process are completed

{

temp=count2;

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

{

if(finish[i]==false) // checking the value of false for each process

{

bool ch = check(i,work);

if(ch==true) // cheking need<work

{

finish[i]=true;

add(i);

ss[count2++]=i; // putting the process in sef sequence

}

if(count2==temp){

goto isnot; //if the value of temp is same as count2 then the loop will end

}

cout<<"";

}

}

}

goto istrue;

isnot:

cout<<"System is not in Safe state \n";

goto end;

istrue:

cout<<"Safe Sequence is \n";

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

cout << ss[i] << " ";

cout<<"\n System is in safe State \n";

end:

cout<<"Thank you";

}

**Safety Algorithm in terms of Operating System and purpose of use:-**

Safety algorithm Is used in Bankers Algorithm to check whether the system is in safe state or not.

If Safety algorithm return false then process has to wait, and if it returns true then only resource is allocated to the process. Bankers and Safety are used in Deadlock avoidance. Safety algorithm generate a safe sequence, the system is in safe state only when there exists a safe sequence. A sequence of processes <*P1, P2, …, Pn*> is a safe sequence for the current allocation state if, for each Pi, the resource requests that Pi can still make, can be satisfied by currently available resources plus resources held by all *Pj*, with *j* < *i*.

That is:If the Pi resource needs are not immediately available, then *Pi* can wait until all *Pj* have finished.

When *Pj* is finished, *Pi* can obtain needed resources, execute, return allocated resources, and terminate .

When *Pi* terminates, *Pi* +1 can obtain its needed resources, and so on.

**1.Available**

It is an **array** of length m. It represents the number of available resources of each type. If Available[j] = k, then there are k instances available, of resource type R(j).

**2. Max**

It is an n x m matrix which represents the maximum number of instances of each resource that a process can request. If Max[i][j] = k, then the process P(i) can request atmost k instances of resource type R(j).

**3.Allocation**It is an n x m matrix which represents the number of resources of each type currently allocated to each process. If Allocation[i][j] = k, then process P(i) is currently allocated k instances of resource type R(j).

**4.Need**

It is an n x m matrix which indicates the remaining resource needs of each process. If Need[i][j] = k, then process P(i) may need k more instances of resource type R(j) to complete its task.

Need[i][j] = Max[i][j] - Allocation [i][j]

**Algorithm:**

1. Let Workand Finish be vectors of lengthm andn, respectively. Initialize:

Work= Available

Finish[i] = *false* fori = 0, 1, …, *n*

2. Find an isuch that

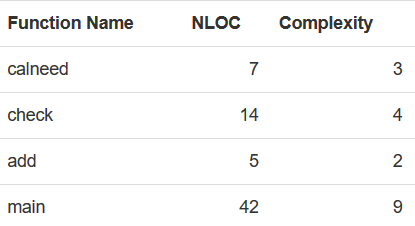
Finish[i] = *false* AND Needi £ Work

If no such iexists, go to step 4

3. Work = Work+ Allocationi Finish[i] = *true*go to step 2

4. If Finish[i] = true for all i, then the system is in a safe stat**e**

**Complexity:**

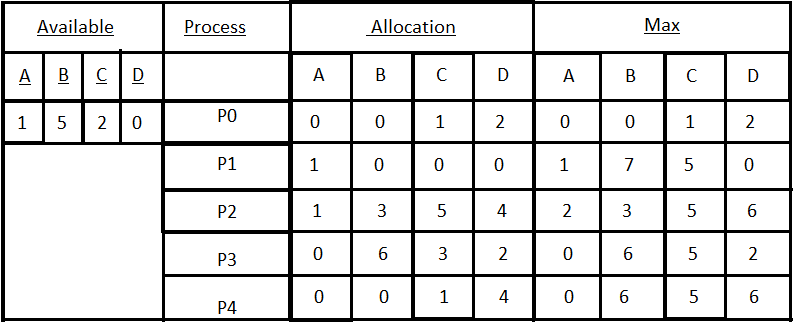
****

**Boundary condition: -**

The code will only check one process at a time therefore will not work in Multi-Access system.

Unnecessary delays in avoiding unsafe state which may lead to deadlock.

**Test case: -**

****