Student Name: Akash Kumar

Student ID: 11712333

Email Address: ak8427916@gmail.com

GitHub Link:

Code:

#include <stdio.h>

const int process=3,resources=3;

void calculate\_safe\_sequence(int [][resources],int [][resources],int [][resources],int \*,int \*);

//calculate need matrix with complexity O(n^2)

void calculate(int need[][resources],int max[][resources],int allocated[][resources]){

for(int i=0;i<process;i++){

for(int j=0;j<resources;j++){

need[i][j]=max[i][j]-allocated[i][j];

}

}

}

//check request can be granted or not and update accordingly with complexity O(n)

void request(int \*processarr,int \*availbale,int \*requestarr,int allocated[][resources],int max[][resources],int index)

{

int need[process][resources];

int ava[resources];

int alloc[process][resources];

//memcpy(ava,availbale,sizeof(availbale));

for(int i=0;i<resources;i++)ava[i]=availbale[i];

for(int i=0;i<process;i++){

for(int j=0;j<resources;j++){

alloc[i][j]=allocated[i][j];

}

}

calculate(need,max,allocated);

int j=0;

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

if(requestarr[j]>need[index][j]||requestarr[j]>availbale[j]){

printf("Request can't be granted for process %d\n",index);

return ;

}

}

if(j==resources){

for(int y=0;y<resources;y++){

//updation

ava[y]-=requestarr[y];

alloc[index][y]+=requestarr[y];

need[index][y]-=requestarr[y];

}

}

printf("Request can be granted for process %d\n",index);

calculate\_safe\_sequence(need,alloc,max,ava,processarr);

}

//calculating safe sequence with updated value with complexity O(n)

void calculate\_safe\_sequence(int need[][resources],int allocated[][resources],int max[][resources],int \*availbale,int \*processarr)

{ //unfinished process

int finish[3]={0,0,0};

int safe[process];

int count=0;

while(count<process){

int found=0;

for(int i=0;i<process;i++){

if(finish[i]==0){

int g=0;

for( g=0;g<resources;g++){

if(need[i][g]>availbale[g]){

break;

}

}

if(g==resources){

for(int y=0;y<resources;y++){

availbale[y]+=allocated[i][y];

}

finish[i]=1;

found=1;

safe[count++]=processarr[i];

}

}

}

if(found==0){

printf("OOPS! No safe sequence exist\n");

return;

}

}

//printing the safe sequenece

printf("safe sequence is:\n");

for(int i=0;i<process;i++)printf("%d ",safe[i]);

}

int main() {

//cin>>process>>resources;

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

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

int allocated[3][3]={

{0, 0, 1},

{3, 2, 0},

{2, 1, 1},

};

int max[3][3]={

{8, 4, 3},

{6, 2, 0},

{3, 3, 3},

};

int number\_of\_request;

scanf("%d",&number\_of\_request);

int requestarr[resources];

while(number\_of\_request--){

int index\_of\_process;

scanf("%d",&index\_of\_process);

for(int j=0;j<resources;j++){

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

}

request(processarr,availbale,requestarr,allocated,max,index\_of\_process);

}

}

Description:

Algorithm for deadlock avoidance to manage the allocation of resources say three namely A, B, and C to three processes P0, P1, and P2

Suppose P0 has 0,0,1 instances

P1 is having 3,2,0 instances

P2 occupies 2,1,1 instances of A,B,C resource respectively.

Also the maximum number of instances required for P0 is 8,4,3 and for P1 is 6,2,0 and finally for P2 there are 3,3,3 instances of resources A,B,C respectively.

Instances are 3 ,2 ,2 for the process

Program to check whether operating system is in a safe state or not in the following independent requests for additional resources in the

Request for resources by P0

Request1: P0 requests 0 instances of A and 0 instances of B and 2 instances of C.

Algortithm:

Given allocation matrix of process , maximum number of instance , available resources

Steps:

1.Calculate need matrix =maximum number – allocation matrix

2.Check request should be granted or not by comparing by need matrix and available resources

3.If 2 is passed then update allocation matrix ,need matrix ,available matrix

4.With updated values calculate safe sequence

Test case:

Case 1:

allocated[3][3]={

{0, 0, 1},

{3, 2, 0},

{2, 1, 1},

};

max[3][3]={

{8, 4, 3},

{6, 2, 0},

{3, 3, 3},

};

processarr[3]={0,1,2};

availbale[3]={3,2,2};

requestarr[3]={0,0,2};

for process P0

Output :

Request can be granted for process 0

OOPS! No safe sequence exists

Case 2:

allocated[3][3]={

{0, 0, 1},

{3, 2, 0},

{2, 1, 1},

};

max[3][3]={

{8, 4, 3},

{6, 2, 0},

{3, 3, 3},

};

processarr[3]={0,1,2};

availbale[3]={3,2,2};

requestarr[3]={2,0,0};

for process P1

Output :

Request can be granted for process 1

safe sequence is:

1 2 0

Case 3:

allocated[3][3]={

{0, 0, 1},

{3, 2, 0},

{2, 1, 3},

};

max[3][3]={

{8, 4, 3},

{6, 2, 0},

{3, 3, 3},

};

processarr[3]={0,1,2};

availbale[3]={3,2,2};

requestarr[3]={0,0,2};

for process P3

Output :

Request can be granted for process 3

safe sequence is:

1 2 0