**Student Name: NaveenChandu.Paritala**

**Student ID: 11805079**

**Email Address: naveenparitala1433@gmail.com**

**GITHUD:** <https://github.com/NaveenChandu03/OS-PROJECT-QUESTIONS>

**Question no.03 Code:**

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

#include<string.h>

#include<pthread.h>

pthread\_mutex\_t lock;

int n;

void \*getuserinput();

void \*showprimenumber();

int main()

{

pthread\_mutex\_init(&lock,NULL);

pthread\_t a\_thread;

pthread\_t b\_thread;

// void \*thread\_result;

pthread\_create(&a\_thread,NULL,getuserinput,NULL);

pthread\_create(&b\_thread,NULL,showprimenumber,NULL);

pthread\_join(a\_thread,NULL);

pthread\_join(b\_thread,NULL);

return 0;

}

void \*getuserinput()

{

pthread\_mutex\_lock(&lock);

printf("Enter the number : ");

scanf("%d",&n);

pthread\_mutex\_unlock(&lock);

}

void \*showprimenumber()

{ int i,j,counter;

pthread\_mutex\_lock(&lock);

if(n>=2)

{

printf("\nALL THE PRIME NUMBERS LESS THAN OR EQUALS TO %d are following : ",n);

for(i = 2; i <= n; i++)

{

counter=0;

for(j = 2; j <= i/2; j++)

{

if(i % j == 0){

counter = 1;

break;

}

}

if(counter==0 && n!= 1)

printf("\n%d ",i);

}

}

else

{

printf("Entered number is less than 2. The smallest prime number is 2...");

}

pthread\_mutex\_unlock(&lock);

}

**Question no.24 Code:**

//program to illustrate Banker's Algorithm

#include<iostream>

using namespace std;

// Number of processes

const int P = 5;

// Number of resources

const int R = 4;

// 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: ";

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

cout << safeSeq[i] << " ";

return true;

}

// Driver code

int main()

{

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

// Available instances of resources

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

// Maximum R that can be allocated

// to processes

int maxm[][R] = {{0, 0, 1, 2},

{1, 7, 5, 0},

{2, 3, 5, 6},

{0, 6, 5, 2},

{0, 6, 5, 6}};

// Resources allocated to processes

int allot[][R] = {{0, 0, 1, 2},

{1, 0, 0, 0},

{1, 3, 5, 4},

{0, 6, 3, 2},

{0, 0, 1, 4}};

// Check system is in safe state or not

isSafe(processes, avail, maxm, allot);

return 0;

}

**Description:**

Description of each line of code is mentioned in the code with complexity.

**Banker’s Safety Algorithm**

The algorithm for finding out whether or not a system is in a safe state can be described as follows:

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

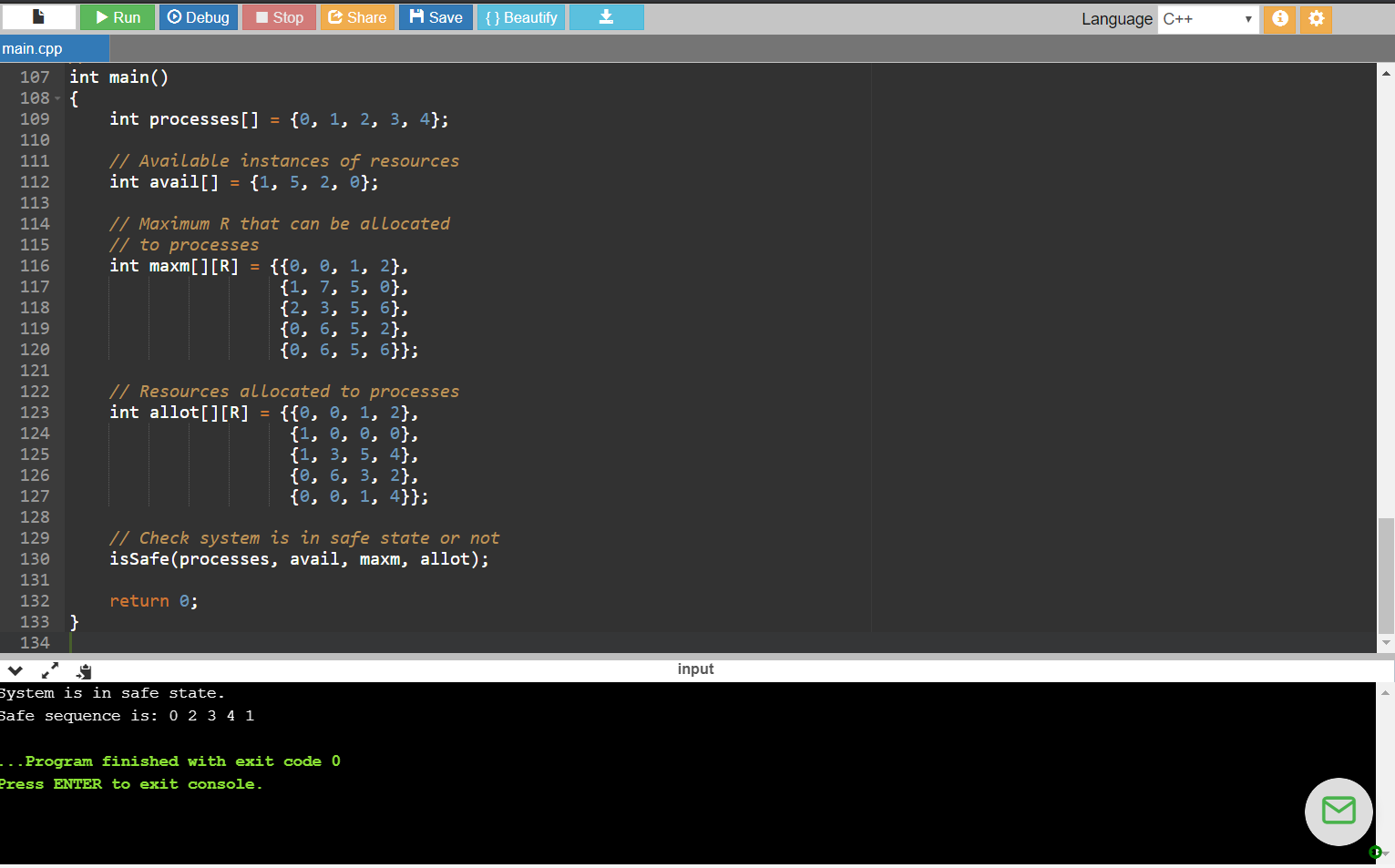
2) Find an i such that both  
a) Finish[i] = false  
b) Needi <= Work  
if no such i exists goto step (4)

3) Work = Work + Allocation[i]  
Finish[i] = true  
goto step (2)

4) if Finish [i] = true for all i  
then the system is in a safe state

**Continued…**

**Code snippet:**



**\*\*\*\*THE END\*\*\*\***