

**School of computer science and engineering**

**Name of the faculty : Dr.Baljit Singh Saini**

Programming assignment

**Course Title** : Operating System

**Course code**: CSE316

**NAME : O.Prashanthreddy**

**REGISTRATION NUMBER : 11805279**

**ROLL NO : 10**

**SECTION : K18VQ**

CA \_3

MAX MARKS : 30

Question:- Write a multithreaded program that implements the banker's algorithm. Create n threads that request and release resources from the bank. The banker will grant the request only if it leaves the system in a safe state. It is important that shared data be safe from concurrent access. To ensure safe access to shared data, you can use mutex locks.

Code:-

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

#include <stdbool.h>

#include <time.h>

int nResources,

nProcesses;

int \*resources;

int \*\*allocated;

int \*\*maxRequired;

int \*\*need;

int \*safeSeq;

int nProcessRan = 0;

pthread\_mutex\_t lockResources;

pthread\_cond\_t condition;

// get safe sequence is there is one else return false

bool getSafeSeq();

// process function

void\* processCode(void\* );

int main(int argc, char\*\* argv) {

srand(time(NULL));

printf("\nNumber of processes? ");

scanf("%d", &nProcesses);

printf("\nNumber of resources? ");

scanf("%d", &nResources);

resources = (int \*)malloc(nResources \* sizeof(\*resources));

printf("\nCurrently Available resources (R1 R2 ...)? ");

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

scanf("%d", &resources[i]);

allocated = (int \*\*)malloc(nProcesses \* sizeof(\*allocated));

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

allocated[i] = (int \*)malloc(nResources \* sizeof(\*\*allocated));

maxRequired = (int \*\*)malloc(nProcesses \* sizeof(\*maxRequired));

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

maxRequired[i] = (int \*)malloc(nResources \* sizeof(\*\*maxRequired));

printf("\n");

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

printf("\nResource allocated to process %d (R1 R2 ...)? ", i+1);

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

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

}

printf("\n");

maximum required resources

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

printf("\nMaximum resource required by process %d (R1 R2 ...)? ", i+1);

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

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

}

printf("\n");

// calculate need matrix

need = (int \*\*)malloc(nProcesses \* sizeof(\*need));

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

need[i] = (int \*)malloc(nResources \* sizeof(\*\*need));

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

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

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

// get safe sequence

safeSeq = (int \*)malloc(nProcesses \* sizeof(\*safeSeq));

for(int i=0; i<nProcesses; i++) safeSeq[i] = -1;

if(!getSafeSeq()) {

printf("\nUnsafe State! The processes leads the system to a unsafe state.\n\n");

exit(-1);

}

printf("\n\nSafe Sequence Found : ");

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

printf("%-3d", safeSeq[i]+1);

}

printf("\nExecuting Processes...\n\n");

sleep(1);

// run threads

pthread\_t processes[nProcesses];

pthread\_attr\_t attr;

pthread\_attr\_init(&attr);

int processNumber[nProcesses];

for(int i=0; i<nProcesses; i++) processNumber[i] = i;

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

pthread\_create(&processes[i], &attr, processCode, (void \*)(&processNumber[i]));

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

pthread\_join(processes[i], NULL);

printf("\nAll Processes Finished\n");

// free resources

free(resources);

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

free(allocated[i]);

free(maxRequired[i]);

free(need[i]);

}

free(allocated);

free(maxRequired);

free(need);

free(safeSeq);

}

bool getSafeSeq() {

// get safe sequence

int tempRes[nResources];

for(int i=0; i<nResources; i++) tempRes[i] = resources[i];

bool finished[nProcesses];

for(int i=0; i<nProcesses; i++) finished[i] = false;

int nfinished=0;

while(nfinished < nProcesses) {

bool safe = false;

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

if(!finished[i]) {

bool possible = true;

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

if(need[i][j] > tempRes[j]) {

possible = false;

break;

}

if(possible) {

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

tempRes[j] += allocated[i][j];

safeSeq[nfinished] = i;

finished[i] = true;

++nfinished;

safe = true;

}

}

}

if(!safe) {

for(int k=0; k<nProcesses; k++) safeSeq[k] = -1;

return false; // no safe sequence found

}

}

return true; // safe sequence found

}

// process code

void\* processCode(void \*arg) {

int p = \*((int \*) arg);

// lock resources

pthread\_mutex\_lock(&lockResources);

// condition check

while(p != safeSeq[nProcessRan])

pthread\_cond\_wait(&condition, &lockResources);

// process

printf("\n--> Process %d", p+1);

printf("\n\tAllocated : ");

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

printf("%3d", allocated[p][i]);

printf("\n\tNeeded : ");

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

printf("%3d", need[p][i]);

printf("\n\tAvailable : ");

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

printf("%3d", resources[i]);

printf("\n"); sleep(1);

printf("\tResource Allocated!");

printf("\n"); sleep(1);

printf("\tProcess Code Running...");

printf("\n"); sleep(rand()%3 + 2); // process code

printf("\tProcess Code Completed...");

printf("\n"); sleep(1);

printf("\tProcess Releasing Resource...");

printf("\n"); sleep(1);

printf("\tResource Released!");

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

resources[i] += allocated[p][i];

printf("\n\tNow Available : ");

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

printf("%3d", resources[i]);

printf("\n\n");

sleep(1);

// condition broadcast

nProcessRan++;

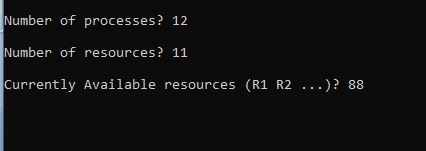
pthread\_cond\_broadcast(&condition);

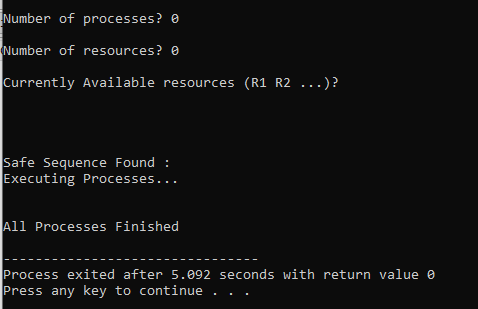
pthread\_mutex\_unlock(&lockResources);

pthread\_exit(NULL);

}

OUTPUT:





2.Explain the boundary conditions of the implemented code.

\*As per question we are bounded to take the n number of processes and n number of resources.

\* Given the Currently Available resources (R1 R2 ...) .

\* If we keep no of processes = 0 and no of resources = 0. It will get all processes finished.