Q-1

b.

//AU1940041 Deepang Desai

#include <stdlib.h>

#include <stdio.h>

#include <pthread.h>

pthread\_cond\_t cond1;

pthread\_cond\_t cond2;

pthread\_cond\_t cond3;

pthread\_mutex\_t mutex;

void\* Pattern1() {

char c;

for (c = 'A'; c <= 'Z'; ++c){

pthread\_mutex\_lock(&mutex);

printf("%c ", c);

pthread\_mutex\_unlock(&mutex);

pthread\_cond\_signal(&cond2);

pthread\_cond\_wait(&cond1,&mutex);

}

}

void\* Pattern2() {

int i;

for(i = 1; i <= 26; i++){

pthread\_mutex\_lock(&mutex);

pthread\_cond\_wait(&cond2,&mutex);

printf("%d ",i);

pthread\_mutex\_unlock(&mutex);

pthread\_cond\_signal(&cond3);

}

}

void\* Pattern3() {

char c;

for (c = 'a'; c <= 'z'; ++c){

pthread\_mutex\_lock(&mutex);

pthread\_cond\_wait(&cond3,&mutex);

printf("%c ", c);

pthread\_mutex\_unlock(&mutex);

pthread\_cond\_signal(&cond1);

}

}

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

pthread\_t p1, p2, p3;

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

pthread\_cond\_init(&cond1, NULL);

pthread\_cond\_init(&cond2, NULL);

pthread\_cond\_init(&cond3, NULL);

if (pthread\_create(&p1, NULL, &Pattern1, NULL) != 0) {

return 1;

}

if (pthread\_create(&p2, NULL, &Pattern2, NULL) != 0) {

return 2;

}

if (pthread\_create(&p3, NULL, &Pattern3, NULL) != 0) {

return 3;

}

if (pthread\_join(p1, NULL) != 0) {

return 4;

}

if (pthread\_join(p2, NULL) != 0) {

return 5;

}

if (pthread\_join(p3, NULL) != 0) {

return 6;

}

pthread\_mutex\_destroy(&mutex);

pthread\_cond\_destroy(&cond1);

pthread\_cond\_destroy(&cond2);

pthread\_cond\_destroy(&cond3);

return 0;

}

Q-2

b.

//AU1940041 Deepang Desai

#include<bits/stdc++.h>

using namespace std;

int size;

vector<pair<int, int>> arr[100000];

map<int, int> mp;

void Buddy(int mem)

{

int n = ceil(log(mem) / log(2));

size = n + 1;

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

arr[i].clear();

arr[n].push\_back(make\_pair(0, mem - 1));

}

void allocate(int s)

{

int x = ceil(log(s) / log(2));

if (arr[x].size() > 0)

{

pair<int, int> temp = arr[x][0];

arr[x].erase(arr[x].begin());

cout << "Memory successfully allocated \n" ;

mp[temp.first] = temp.second -temp.first + 1;

}

else

{

int i;

for(i = x + 1; i < size; i++)

{

if (arr[i].size() != 0)

break;

}

if (i == size)

{

cout << "Sorry, failed to allocate memory\n";

}

else

{

pair<int, int> temp;

temp = arr[i][0];

arr[i].erase(arr[i].begin());

i--;

for(;i >= x; i--)

{

pair<int, int> pair1, pair2;

pair1 = make\_pair(temp.first,temp.first +(temp.second -temp.first) / 2);

pair2 = make\_pair(temp.first +(temp.second -temp.first + 1) / 2,temp.second);

arr[i].push\_back(pair1);

arr[i].push\_back(pair2);

temp = arr[i][0];

arr[i].erase(arr[i].begin());

}

cout << "Memory suceesfully allocated \n";

mp[temp.first] = temp.second -temp.first + 1;

}

}

}

void deallocate(int id)

{

if(mp.find(id) == mp.end())

{

cout << "Sorry, invalid free request\n";

return;

}

int n = ceil(log(mp[id]) / log(2));

int i, b\_num, b\_add;

arr[n].push\_back(make\_pair(id,id + pow(2, n) - 1));

cout << "Memory freed succesfully\n" ;

b\_num = id / mp[id];

if (b\_num % 2 != 0)

b\_add = id - pow(2, n);

else

b\_add = id + pow(2, n);

for(i = 0; i < arr[n].size(); i++)

{

if (arr[n][i].first == b\_add)

{

if (b\_num % 2 == 0)

{

arr[n + 1].push\_back(make\_pair(id,id + 2 \* (pow(2, n) - 1)));

cout << "Coalescing of buddies done succesfully\n";

}

else

{

arr[n + 1].push\_back(make\_pair(

b\_add, b\_add +

2 \* (pow(2, n))));

cout << "Coalescing of buddies done succesfully \n";

}

arr[n].erase(arr[n].begin() + i);

arr[n].erase(arr[n].begin() +

arr[n].size() - 1);

break;

}

}

mp.erase(id);

}

int main()

{

Buddy(1024);

allocate(64);

allocate(128);

allocate(255);

allocate(256);

deallocate(128);

deallocate(16);

deallocate(32);

return 0;

}

Q-3

//AU1940041 Deepang Desai

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <pthread.h>

#include <unistd.h>

#include <time.h>

#include <semaphore.h>

sem\_t seme;

sem\_t semf;

pthread\_mutex\_t mutexBuffer;

int buffer[10];

int count = 0;

void\* producer(void\* args) {

while (1) {

int x = rand();

sleep(1);

sem\_wait(&seme);

pthread\_mutex\_lock(&mutexBuffer);

buffer[count] = x;

count++;

pthread\_mutex\_unlock(&mutexBuffer);

sem\_post(&semf);

}

}

void\* consumer(void\* args) {

while (1) {

int y;

sem\_wait(&semf);

pthread\_mutex\_lock(&mutexBuffer);

y = buffer[count - 1];

count--;

pthread\_mutex\_unlock(&mutexBuffer);

sem\_post(&seme);

// Consume

printf("Got %d\n", y);

sleep(1);

}

}

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

srand(time(NULL));

pthread\_t th[8];

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

sem\_init(&seme, 0, 10);

sem\_init(&semf, 0, 0);

int i;

for (i = 0; i < 8; i++) {

if (i > 0) {

if (pthread\_create(&th[i], NULL, &producer, NULL) != 0) {

perror("Failed to create thread");

}

} else {

if (pthread\_create(&th[i], NULL, &consumer, NULL) != 0) {

perror("Failed to create thread");

}

}

}

for (i = 0; i < 8; i++) {

if (pthread\_join(th[i], NULL) != 0) {

perror("Failed to join thread");

}

}

sem\_destroy(&seme);

sem\_destroy(&semf);

pthread\_mutex\_destroy(&mutexBuffer);

return 0;

}