Assignment 7 -

// Header file include

#include <bits/stdc++.h>

#include <pthread.h>

#include <unistd.h>

using namespace std;

#define N 5

#define THINKING 2

#define HUNGRY 1

#define EATING 0

#define LEFT (phnum + 4) % N

#define RIGHT (phnum + 1) % N

// Philosopher index

int phil[N];

int times = 10;

class monitor {

// state of the philosopher

int state[N];

// Philosopher condition variable

pthread\_cond\_t phcond[N];

// mutex variable for synchronization

pthread\_mutex\_t condLock;

public:

// Test for the desired condition

// i.e. Left and Right philosopher are not reading

void test(int phnum)

{

if (state[(phnum + 1) % 5] != EATING

and state[(phnum + 4) % 5] != EATING

and state[phnum] == HUNGRY) {

state[phnum] = EATING;

pthread\_cond\_signal(&phcond[phnum]);

}

}

// Take Fork function

void take\_fork(int phnum)

{

pthread\_mutex\_lock(&condLock);

// Indicates it is hungry

state[phnum] = HUNGRY;

// test for condition

test(phnum);

// If unable to eat.. wait for the signal

if (state[phnum] != EATING) {

pthread\_cond\_wait(&phcond[phnum], &condLock);

}

cout << "Philosopher " << phnum << " is Eating"<< endl;

pthread\_mutex\_unlock(&condLock);

}

// Put Fork function

void put\_fork(int phnum)

{

pthread\_mutex\_lock(&condLock);

// Indicates that I am thinking

state[phnum] = THINKING;

test(RIGHT);

test(LEFT);

pthread\_mutex\_unlock(&condLock);

}

// constructor

monitor()

{

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

state[i] = THINKING;

}

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

pthread\_cond\_init(&phcond[i], NULL);

}

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

}

// destructor

~monitor()

{

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

pthread\_cond\_destroy(&phcond[i]);

}

pthread\_mutex\_destroy(&condLock);

}

}

// Global Object of the monitor

phil\_object;

void\* philosopher(void\* arg)

{

int c = 0;

while (c < times) {

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

sleep(1);

phil\_object.take\_fork(i);

sleep(0.5);

phil\_object.put\_fork(i);

c++;

}

}

int main() {

// Declaration...

pthread\_t thread\_id[N];

pthread\_attr\_t attr;

// Initialization...

pthread\_attr\_init(&attr);

pthread\_attr\_setdetachstate(&attr,

PTHREAD\_CREATE\_JOINABLE);

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

phil[i] = i;

}

// Creating...

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

pthread\_create(&thread\_id[i], &attr, philosopher,

&phil[i]);

cout << "Philosopher " << i + 1 << " is thinking..."

<< endl;

}

// Joining....

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

pthread\_join(thread\_id[i], NULL);

}

// Destroying

pthread\_attr\_destroy(&attr);

pthread\_exit(NULL);

return 0;

}

Assignment 8 –

// g++ -pthread program\_name.cpp

// CPP Program to multiply two matrix using pthreads

#include <bits/stdc++.h>

using namespace std;

// maximum size of matrix

#define MAX 2

// maximum number of threads

#define MAX\_THREAD 2

int matA[MAX][MAX];

int matB[MAX][MAX];

int matC[MAX][MAX];

int step\_i = 0;

void\* multi(void\* arg)

{

int i = step\_i++; //i denotes row number of resultant matC

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

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

matC[i][j] += matA[i][k] \* matB[k][j];

}

}

// Driver Code

int main()

{

// Generating random values in matA and matB

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

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

matA[i][j] = rand() % 10;

matB[i][j] = rand() % 10;

}

}

// Displaying matA

cout << endl

<< "Matrix A" << endl;

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

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

cout << matA[i][j] << " ";

cout << endl;

}

// Displaying matB

cout << endl

<< "Matrix B" << endl;

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

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

cout << matB[i][j] << " ";

cout << endl;

}

// declaring four threads

pthread\_t threads[MAX\_THREAD];

// Creating four threads, each evaluating its own part

for (int i = 0; i < MAX\_THREAD; i++) {

int\* p;

pthread\_create(&threads[i], NULL, multi, (void\*)(p));

}

// joining and waiting for all threads to complete

for (int i = 0; i < MAX\_THREAD; i++)

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

// Displaying the result matrix

cout << endl

<< "Multiplication of A and B" << endl;

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

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

cout << matC[i][j] << " ";

cout << endl;

}

return 0;

}