**PROGRAMING ASSIGNMENT – 02**

**The solution to the programming Assignment has been developed in c++ language and compiled and run on linux.**

**Source Code:**

#include <iostream>

#include <pthread.h>

#include <unistd.h>

#include <mutex>

#include <thread>

using namespace std;

class Program {

private:

// no. of readers

int rcnt;

// no. of writers

int wcnt;

// no. of readers waiting

int waitr;

// no. of writers waiting

int waitw;

// condition variable to check whether reader can read

pthread\_cond\_t canread;

// condition variable to check whether writer can write

pthread\_cond\_t canwrite;

// mutex for synchronisation

pthread\_mutex\_t condlock;

public:

Program()

{

rcnt = 0;

wcnt = 0;

waitr = 0;

waitw = 0;

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

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

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

}

//Probability function

int probability()

{

srand(time(0));

return (rand()%((20+1-0)+0));

}

// mutex provide synchronisation so that no other thread

// can change the value of data

void beginread(int i)

{

pthread\_mutex\_lock(&condlock);

cout << "\nreader " << i << " acquired mutex lock for reading\n";

// if there are active or waiting writers

if (wcnt == 1 || waitw > 0) {

waitr++;

cout << "reader " << i << " cannot read writer process has acquired the lock.\n";

pthread\_cond\_wait(&canread, &condlock);

waitr--;

}

// else reader reads.

rcnt++;

cout << "reader " << i << " is reading\n";

pthread\_mutex\_unlock(&condlock);

cout << "reader " << i << " released mutex lock\n";

pthread\_cond\_broadcast(&canread);

}

void endread(int i)

{

// if there are no readers left then writer can write.

pthread\_mutex\_lock(&condlock);

cout << "\nreader " << i << " acquired mutex lock\n";

if (--rcnt == 0)

pthread\_cond\_signal(&canwrite);

cout << "reader " << i << " finished reading now other process can perform.\n";

pthread\_mutex\_unlock(&condlock);

cout << "reader " << i << " released mutex lock\n";

}

void beginwrite(int i)

{

pthread\_mutex\_lock(&condlock);

cout << "\nwriter " << i << " acquired mutex lock\n";

// a writer can enter when there are no active

// or waiting readers or other writer

if (wcnt == 1 || rcnt > 0) {

++waitw;

pthread\_cond\_wait(&canwrite, &condlock);

--waitw;

}

wcnt = 1;

cout << "writer " << i << " is writing\n";

pthread\_mutex\_unlock(&condlock);

cout << "writer " << i << " released mutex lock\n";

}

void endwrite(int i)

{

pthread\_mutex\_lock(&condlock);

cout << "\nwriter " << i << " acquired mutex lock\n";

wcnt = 0;

// if any readers are waiting, resources are unblocked

if (waitr > 0)

{

pthread\_cond\_signal(&canread);

cout << "writer " << i << " finsished writing\n";

}

else

{

pthread\_cond\_signal(&canwrite);

cout << "writer " << i << " finsished writing\n";

}

pthread\_mutex\_unlock(&condlock);

cout << "writer " << i << " released mutex lock\n";

}

}

// global object of Program class

P;

void\* reader(void\* id)

{

int c = 0;

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

// each reader attempts to read 3 times

while (c < 3) {

usleep(2); // using sleep fuction

int prob = P.probability();

if(prob>=5)

{

P.beginread(i);

P.endread(i);

c++;

}

}

return 0;

}

void\* writer(void\* id)

{

int c = 0;

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

// each writer attempts to write 3 times

while (c < 3) {

usleep(2); // using sleep fuction

int prob = P.probability();

if(prob>=5)

{

P.beginwrite(i);

P.endwrite(i);

c++;

}

}

return 0;

}

int main()

{

pthread\_t r[5], w[2]; // 2 writer threads and 5 reader threads

int id[5];

int pid[2];

for (int i = 0; i < 5; i++) {

id[i] = i;

// creating threads which execute reader function

pthread\_create(&r[i], NULL, &reader, &id[i]);

if(i < 2)

{

pid[i] = i;

// creating threads which execute writer function

pthread\_create(&w[i], NULL, &writer, &pid[i]);

}

}

for (int i = 0; i < 5; i++) {

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

if(i < 2)

{

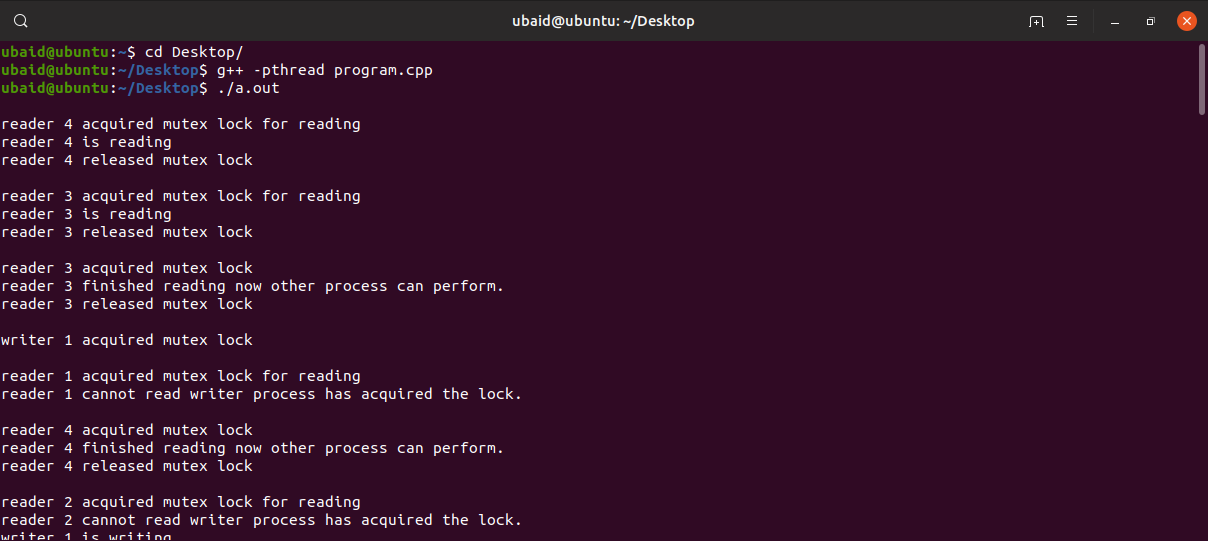
pthread\_join(w[i], NULL);

}

}

}

**Result:**

****