

**School of computer science and engineering**

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

Programming assignment

**Course Title** : Operating System

**Course code**: CSE316

**NAME : B. Sai Krishna**

**REGISTRATION NUMBER : 11805293**

**ROLL NO : 08**

**SECTION : K18VQ**

CA \_3

MAX MARKS : 30

Question:- Write a multi-threaded C program that gives readers priority over writers concerning a shared (global) variable. Essentially, if any readers are waiting, then they have priority over writer threads -- writers can only write when there are no readers. This program should adhere to the following constraints:

Multiple readers/writers must be supported (5 of each is fine)

Readers must read the shared variable X number of times

Writers must write the shared variable X number of times

Readers must print:

The value read

The number of readers present when value is read

Writers must print:

The written value

The number of readers present were when value is written (should be 0)

Before a reader/writer attempts to access the shared variable it should wait some random amount of time.

Code:-

#include <iostream>

#include <condition\_variable>

#include <thread>

#include <vector>

#include <random>

#include <mutex>

int reader = 0;

int c = 0;

const int X = 6;

std::mutex m;

std::condition\_variable reader\_cond;

std::condition\_variable writer\_cond;

void read()

{

std::mt19937 rng;

rng.seed(std::random\_device()());

std::uniform\_int\_distribution<std::mt19937::result\_type> dist(0, 20);

std::unique\_lock<std::mutex> lk(m, std::defer\_lock);

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

std::this\_thread::sleep\_for(std::chrono::milliseconds(dist(rng)));

lk.lock();

if(reader == -1) {

reader\_cond.wait(lk, [](){ return reader!= -1; });

}

reader++;

lk.unlock();

std::cout << "read value: " << c << ", number of readers: " << reader << std::endl;

lk.lock();

reader--;

if(reader == 0){

writer\_cond.notify\_all();

}

lk.unlock();

}

}

void write() {

std::mt19937 rng;

rng.seed(std::random\_device()());

std::uniform\_int\_distribution<std::mt19937::result\_type> dist(0, 20);

std::unique\_lock<std::mutex> lk(m, std::defer\_lock);

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

std::this\_thread::sleep\_for(std::chrono::milliseconds(dist(rng)));

lk.lock();

if(reader > 0) {

writer\_cond.wait(lk, [](){ return reader == 0; });

}

readers\_count = -1;

lk.unlock();

c++;

std::cout << "written value: " << c << ", number of readers: " << reader << std::endl;

lk.lock();

reader = 0;

reader\_cond.notify\_all();

writer\_cond.notify\_all();

lk.unlock();

}

}

int main() {

const int NUM\_READERS = 5;

const int NUM\_WRITERS = 6;

std:: cout << std::thread::hardware\_concurrency() << std::endl;

std::vector<std::thread> threads;

for(int i = 0; i < NUM\_READERS; i++) {

threads.push\_back(std::thread{read});

}

for(int i = 0; i < NUM\_WRITERS; i++) {

threads.push\_back(std::thread{write});

}

for(int i = 0; i < NUM\_READERS + NUM\_WRITERS; i++) {

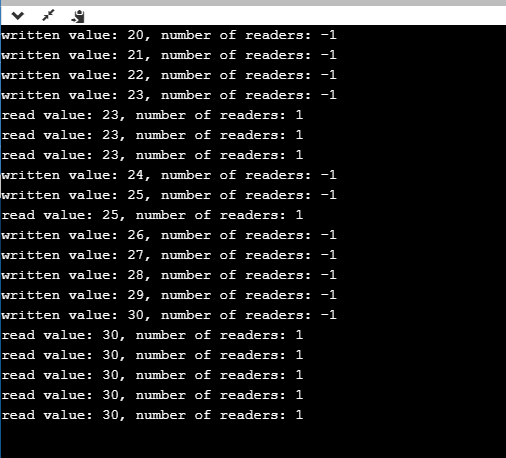
threads[i].join();

}

return 0;

}

OUTPUT:



2.Explain the boundary conditions of the implemented code.

\*As per question we are bounded to take the number of readers has now increased by 1: readcnt++;

\*Reader wants to enter the critical section: wait(mutex);

\*There is atleast onr reader in the critical section.

\*Other readers can enter while this current reader is inside.

3.Explain all the test cases applied on the solution of assigned problem.

Test Case 1:

When the user Enter Number of “WRITER” is Negative.

* No WRITER and READER thread will be Created.
* No read or write operation done.
* Program will be terminated.

Test Case 2:

When the user Enter Number of “READER” is Negtive.

* No WRITER and READER thread will be created.
* No read and write operation done.
* Program eill be terminated.

Test Case 3:

When Number of “WRITER=0” is taken as input

* Only READER thread will be created.
* No WRITER thread is created.
* No updating of the shared variable.

Test Case 4:

When Number of “READER=0” is taken as input.

* Only WRITER thread is created.
* No READER thread is created.
* No reading of the shared variable.