```
/*
 * Fraction.cpp
 *
 * Multiple threads accessing a simple fraction object
 *
 * 2019 C. Scratchley created to work with C++11/14
 *
 * Copyright 2019, School of Engineering Science, SFU, Canada
*/

#include <iostream>
#include <signal.h>
#include <memory>
#include <sched.h>      // sched_yield()
#include <unistd.h>     /* required for sleep() */
#include <stdlib.h>     /* abs() ? */
#include <thread>
#include <mutex>        // std :: mutex.h // mutual exclusion

using namespace std;

mutex consoleMutex;
#define COUT (lock_guard<mutex>(consoleMutex), std::cout)

class fraction
{
private: mutex myMutex;  //good
public:
    unsigned num;
    unsigned denom;

    fraction(unsigned numerator, unsigned denominator) {
        num = numerator;
        denom = denominator;

    };
    void report(char id) {
        unsigned numC = num;
        unsigned denomC = denom;

        cout << id << ": Numerator is " << numC << ". Denominator is " << denomC
             << ".  Floating representation is " << 1.0 * numC / denomC
             << ".  Integer representation is " << numC / denomC
             << endl;
    };
    void invert() {
        int diff = num - denom; // subtract num count from denom count
        if (diff >= 0) {
            sched_yield();
            num -= diff;        // remove from num
            denom += diff;      // and... insert into denom
        }
```

Oct 7th recording

copy mutex

delete.

chapter 3

help to fix the code

where should we put a mutex?

use lock mutex to have only 1 thread running.

either public or private.

//bad, 2 threads may be working on 2 different vars, and mutex will prevent it.

to protect two lines data below

lock_guard guard{myMutex}

unwanted

only

// context switch here b/w threads

make floating type

// lock_guard guard(myMutex)

// 2/3 → 3/2

// lock a mutex

// may cause crash. the program is changing data, jump to other funcs, can cause wrong data.

```
void invert(){
    mutex myMutex; // bad because each caller gets own mutex
}
myMutex.lock();
```

```
        else {
            unsigned adiff = abs(diff);
            denom -= adiff;     // remove from denom
            num += adiff;       // and... insert into num
        }
    };
};
```

mutex ask thread 'B' to wait until 'A' unlock mutex

// unlock the mutex mymutex.unlock();

// washroom anology.
mutex lock → door lock
sched_yield → disability/senior priority.

```
//fraction fraction1(2, 3);
fraction fraction1(3, 6);

void threadFunc(char id) {
//    while (true) {
    for (int i=0; i < 3000; i++) {
        //
        fraction1.invert();
        // possibly some code here with random execution time.
        fraction1.report(id);
//      sched_yield();
    };
};

int main(int argc, char *argv[]) {
    /*  Define behaviour for divide-by-zero */
    //  SIGFPE is for integer divide-by-zero too.
    //  Single Unix Specification defines SIGFPE as "Erroneous arithmetic operation."
    //  https://stackoverflow.com/questions/6121623/catching-exception-divide-by-zero
    std::shared_ptr<void(int)> handler(
        signal(SIGFPE, [](int signum) {throw std::logic_error("FPE"); }),
        [](__sighandler_t f) { signal(SIGFPE, f); });

    fraction1.report('P');

    thread threadA(threadFunc, 'A');
    thread threadB(threadFunc, 'B');
    // sleep(1);
    threadB.join();
    threadA.join();

    cout << "Primary thread ending" << std::endl;
    return EXIT_SUCCESS;
}
```

fraction1 global object

'A', 'B', 'C' thread

// absolute priority higher than 0

// give up processor and allow another thread to use the processor

mutex mymutex; // bad, mutex as data member, make every class has mutex.

// 0

thread (file)

Cxx → C++

```
Mutex myMutex;    //global object
class fraction{
private:
    mutex MyMutex;
};

invert() {
    myMutex; // it will call global or member mutex ?? Global?
}            // fraction object is global
```

- RClick to var → add watch expression to see global var in debugger

::myMutex
↖ scope resolution
↖ global namespace.

std::myMutex
↖ locally s.t in standard.

The short answer to your question is that **futexes are known to be implemented about as efficiently as possible**, while a pthread mutex may or may not be. At minimum, a pthread mutex has overhead associated with determining the type of mutex and futexes do not.
Sep. 17, 2011

https://stackoverflow.com › questions › why-is-a-pthread-...
Why is a pthread mutex considered "slower" than a futex?

linux → search → system monitor

chap 3. p48

```
invert() {
    lock_guard<mutex> guard(myMutex);
    std::lock_guard guard(myMutex);
    return;
}
```

C++17

{myMutex}; // new C++

// constructed : lock the mutex
// don't need to myMutex.lock() and unlock() manually

- when destructor for guard being involve?

- Code Analysis:
  Unticlc to accept dump errors (b/w C++ versions)

* Console:
Jumpo output disorder
B/w insertion, two threads run as same time
Think! How to fix the code?

```
void report(char id) {
{   unsigned numC = num;
    unsigned denomC = denom;

    cout << id << ": Numerator is " << numC << ", Denominator is " << denomC
         << ".  Floating representation is " << 1.0 * numC / denomC
         << ".  Integer representation is " << numC / denomC
         << endl;

};
void invert() {
```

lock_guard {myMutex}

copy

insertion

// 2/3 → 3/2

Soln:

```
#define COUT (lock_guard<mutex>(consoleMutex), std::cout)
```

comma operator

(value at last comma)