Assignment 27-08-2024

1. Please fix the issue in below code.

void bad()

m.lock(); // acquire the mutex

f(); // if f() throws an exception, the mutex is never released

if (!everything\_ok())

return; // early return, the mutex is never released

m.unlock(); // if bad() reaches this statement, the mutex is released

}

Solution: Using lock Guard

To fix this issue, we can use a lock guard, which is a RAII (Resource Acquisition Is Initialization) idiom that ensures the mutex is released when it goes out of scope, regardless of whether an exception is thrown or not.

#include <iostream>

#include <mutex>

#include <stdexcept>

// Mutex and lock guard example

std::mutex m;

void f() {

std::cout << "f() called" << std::endl;

// ... do some work

}

bool everything\_ok() {

// ... check if everything is ok

return true;

}

void good() {

std::lock\_guard<std::mutex> lock(m); // acquire the mutex

f(); // if f() throws an exception, the mutex is released

if (!everything\_ok()) {

return; // early return, the mutex is released

}

// no need to explicitly unlock the mutex

}

int main() {

good();

std::cout << "Back in main()" << std::endl;

return 0;

}

2. Write multi thread program on fetch\_add, fetch\_or, fetch\_and, fecth\_xor

#include <iostream>

#include <atomic>

#include <thread>

std::atomic<int> x(0); // shared atomic variable

void thread\_func(int id) {

for (int i = 0; i < 10000; ++i) {

switch (id) {

case 0:

x.fetch\_add(1); // increment x

break;

case 1:

x.fetch\_or(1); // set x to x | 1

break;

case 2:

x.fetch\_and(~1); // set x to x & ~1

break;

case 3:

x.fetch\_xor(1); // set x to x ^ 1

break;

}

}

}

int main() {

std::thread t1(thread\_func, 0);

std::thread t2(thread\_func, 1);

std::thread t3(thread\_func, 2);

std::thread t4(thread\_func, 3);

t1.join();

t2.join();

t3.join();

t4.join();

std::cout << "Final value of x: " << x << std::endl;

return 0;

}