With C++14 came reader-writer locks. The idea is straightforward and promising. Arbitrary reading threads can access the critical region at the same time, but only one thread is allowed to write.

Minimized bottleneck

Reader-writer locks do not solve the fundamental problem - threads competing for access to a critical region. But reader-writer locks help a lot - to minimize the bottleneck. Let's have an example.

// readerWriterLock.cpp

#include <iostream>

#include <map>

#include <shared\_mutex>

#include <string>

#include <thread>

std::map<std::string,int> teleBook{{"Dijkstra",1972},{"Scott",1976},{"Ritchie",1983}};

std::shared\_timed\_mutex teleBookMutex;

void addToTeleBook(const std::string& na, int tele){

std::lock\_guard<std::shared\_timed\_mutex> writerLock(teleBookMutex);

std::cout << "\nSTARTING UPDATE " << na;

std::this\_thread::sleep\_for(std::chrono::milliseconds(500));

teleBook[na]= tele;

std::cout << " ... ENDING UPDATE " << na << std::endl;

}

void printNumber(const std::string& na){

std::shared\_lock<std::shared\_timed\_mutex> readerLock(teleBookMutex);

std::cout << na << ": " << teleBook[na];

}

int main(){

std::cout << std::endl;

std::thread reader1([]{ printNumber("Scott"); });

std::thread reader2([]{ printNumber("Ritchie"); });

std::thread w1([]{ addToTeleBook("Scott",1968); });

std::thread reader3([]{ printNumber("Dijkstra"); });

std::thread reader4([]{ printNumber("Scott"); });

std::thread w2([]{ addToTeleBook("Bjarne",1965); });

std::thread reader5([]{ printNumber("Scott"); });

std::thread reader6([]{ printNumber("Ritchie"); });

std::thread reader7([]{ printNumber("Scott"); });

std::thread reader8([]{ printNumber("Bjarne"); });

reader1.join();

reader2.join();

reader3.join();

reader4.join();

reader5.join();

reader6.join();

reader7.join();

reader8.join();

w1.join();

w2.join();

std::cout << std::endl;

std::cout << "\nThe new telephone book" << std::endl;

for (auto teleIt: teleBook){

std::cout << teleIt.first << ": " << teleIt.second << std::endl;

}

std::cout << std::endl;

}

The screenshot  shows that the output of the reading threads overlaps, while the writing thread are executed one after the other. It means that the reading operations are performed at the same time.