**Лекция 18.11.2022**

#include <queue>  
#include <memory>  
#include <mutex>  
#include <condition\_variable>  
#include <iostream>  
#include <fstream>  
  
*template*<*typename* T>  
*class* threadsafe\_queue  
{  
*private*:  
 *mutable* std::mutex mut;  
 std::queue<T> data\_queue;  
 std::condition\_variable data\_cond;  
*public*:  
 threadsafe\_queue()= *default*;  
  
 threadsafe\_queue(threadsafe\_queue *const*& other) {  
 std::lock\_guard<std::mutex> lk(other.mut);  
 data\_queue = other.data\_queue;  
 }  
  
 *void* push(T new\_value) {  
 std::lock\_guard<std::mutex> lk(mut);  
 data\_queue.push(new\_value);  
 data\_cond.notify\_one();  
 }  
  
 *void* wait\_and\_pop(T& value) {  
 std::unique\_lock<std::mutex> lk(mut);  
 data\_cond.wait(lk, [*this*]{*return* !data\_queue.empty();});  
 value = data\_queue.pop();  
 }  
 std::shared\_ptr<T> wait\_and\_pop(){  
 std::unique\_lock<std::mutex> lk(mut);  
 data\_cond.wait(lk,[*this*]{*return* !data\_queue.empty();});  
 std::shared\_ptr<T> res(std::make\_shared<T>(data\_queue.front()));  
 data\_queue.pop();  
 *return* res;  
 }  
  
 *bool* try\_pop(T& value){  
 std::lock\_guard<std::mutex> lk(mut);  
 *if*(data\_queue.empty()){  
 *return false*;  
 }  
 value=data\_queue.front();  
 data\_queue.pop();  
 *return true*;  
 }  
  
 std::shared\_ptr<T> try\_pop(){  
 std::lock\_guard<std::mutex> lk(mut);  
 *if*(data\_queue.empty()){  
 *return* std::shared\_ptr<T>();  
 }  
 std::shared\_ptr<T> res(std::make\_shared<T>(data\_queue.front()));  
 data\_queue.pop();  
 *return* res;  
 }  
 *bool* empty() *const*{  
 std::lock\_guard<std::mutex> lk(mut);  
 *return* data\_queue.empty();  
 }  
};  
  
  
*using namespace* std::chrono\_literals;  
  
*class* Consumer {  
*private*:  
 *static* std::ofstream fout;  
*public*:  
 *void operator*()(threadsafe\_queue<std::string>& request\_queue, std::mutex& mutex, *bool*& finished) {  
 *while* (!finished) {  
 fout << request\_queue.try\_pop() << " ";  
 }  
 }  
};  
  
*class* Producer {  
*private*:  
 *static* std::ifstream fin;  
 std::string buffer\_;  
*public*:  
 *void operator*()(threadsafe\_queue<std::string>& request\_queue, std::mutex& mutex, *bool*& finished) {  
 *while*(!fin.eof()){  
 {  
 fin >> buffer\_;  
 request\_queue.push(buffer\_);  
 }  
 }  
 {  
 finished = *true*;  
 }  
 }  
};  
  
std::ifstream Producer::fin = std::ifstream("input.txt");  
std::ofstream Consumer::fout = std::ofstream("output.txt");  
  
*int* main() {  
 *auto* start = std::chrono::high\_resolution\_clock::now();  
 std::mutex mut;  
 threadsafe\_queue<std::string> request\_queue;  
 *bool* finished = *false*;  
 std::thread t1(Consumer(), std::ref(request\_queue), std::ref(mut), std::ref(finished));  
 std::thread t2(Producer(), std::ref(request\_queue), std::ref(mut), std::ref(finished));  
 t1.join();  
 t2.join();  
 std::cout << "finished, time is " << std::chrono::duration\_cast<std::chrono::microseconds>(std::chrono::high\_resolution\_clock::now()- start).count() << " mcs\n";  
}

Результаты вычислений:

