#include <iostream>

#include <thread>

#include <mutex>

#include <stack>

#include <vector>

#include <Windows.h>

#include <algorithm>

using std::cin;

using std::cout;

using TInfo = std::vector<int>;

const size\_t COUNT = 10;

CRITICAL\_SECTION cs; //Только для печати

std::ostream& operator<<(std::ostream& out, TInfo obj)

{

//EnterCriticalSection(&cs);

for (int i = 0; i < obj.size() - 1; ++i)

out << obj[i] << ' ';

out << obj[obj.size() - 1];

//LeaveCriticalSection(&cs);

return out;

}

class ThreadSafeStack

{

private:

// мьютекс для обеспечения эксклюзивного доступа

// к элементам контейнера

std::mutex mutex;

std::stack<TInfo> stack;

public:

void push(TInfo elem, int ID)

{

std::lock\_guard<std::mutex> locker(mutex);

stack.push(elem);

EnterCriticalSection(&cs);

std::cout << 'P' << ID << " -> " << elem << '\n';

LeaveCriticalSection(&cs);

}

bool try\_pop(TInfo& elem, int ID)

{

bool result = false;

std::lock\_guard<std::mutex> locker(mutex);

if (!stack.empty())

{

result = true;

elem = stack.top();

stack.pop();

EnterCriticalSection(&cs);

std::cout << 'C' << ID << " <- " << elem << '\n';

LeaveCriticalSection(&cs);

}

else

{

EnterCriticalSection(&cs);

std::cout << 'C' << ID << " sleep\n";

std::this\_thread::sleep\_for(std::chrono::milliseconds(5));

LeaveCriticalSection(&cs);

}

return result;

}

bool empty()

{

return stack.empty();

}

};

ThreadSafeStack TSS;

//мьютекс для метода wait()

//std::mutex mut;

HANDLE hEvent;

volatile long volume\_work\_producer = 10;

volatile long volume\_work\_consumer = 10;

void task\_producer(int ID)

{

//EnterCriticalSection(&cs);

//cout << 'P' << ID << " начал производить\n";

//LeaveCriticalSection(&cs);

// пока есть работа у потока-производителя

while (\_InterlockedExchangeAdd(&volume\_work\_producer, -1) > 0)

{

// формируется элемент для последующей обработки

// потоком-потребителем

TInfo elem;

for (size\_t i = 0; i < COUNT; i++)

elem.push\_back(rand() % 201 - 100);

//std::this\_thread::sleep\_for(std::chrono::milliseconds(2));

TSS.push(elem, ID);

// поток-производитель передает сигнал,

// что в контейнере появился элемент для обработки

//cout << 'P' << ID << " поставил событие\n";

SetEvent(hEvent);

}

}

void task\_consumer(int ID)

{

// пока есть работа у потока-потребителя

while (\_InterlockedExchangeAdd(&volume\_work\_consumer, -1) > 0)

{

TInfo elem;

//std::unique\_lock<std::mutex> locker(mut);

if (TSS.empty())

{

//cout << 'C' << ID << " начал ждать\n";

WaitForSingleObject(hEvent, 10);

}

//cout << 'C' << ID << " закончил ждать\n";

if (TSS.try\_pop(elem, ID))

{

// если элемент извлечен из контейнера,

//то он обрабатывается потоком-потребителем

//std::this\_thread::sleep\_for(std::chrono::milliseconds(5));

std::sort(elem.begin(), elem.end());

EnterCriticalSection(&cs);

cout << 'C' << ID << " посортировал: " << elem << std::endl;

LeaveCriticalSection(&cs);

}

else

{

// работа не выполнена, объем работы потребителя

// возвращается в исходное состояние

\_InterlockedExchangeAdd(&volume\_work\_consumer, 1);

SetEvent(hEvent);

}

//cout << 'C' << ID << " поставил событие\n";

//SetEvent(hEvent);

}

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand(GetTickCount());

//(атрибут защиты, тип сброса TRUE - ручной, начальное состояние TRUE - сигнальное, имя обьекта)

hEvent = CreateEvent(nullptr, false, false, nullptr);

InitializeCriticalSection(&cs); //mutex внутри стека даёт атомарный доступ печати стеку, а не всей программе

// создание потоков:

// 2 потока-производителя (producer);

// 3 потока-потребителя (consumer);

std::thread worker[5];

for (int i = 0; i < 5; i++)

if (i < 2)

worker[i] = std::thread(task\_producer, i);

else

worker[i] = std::thread(task\_consumer, i);

for (int i = 0; i < 5; i++)

worker[i].join();

DeleteCriticalSection(&cs);

std::cin.get();

return 0;

}