Белорусский государственный университет

информатики и радиоэлектроники

Кафедра ПОИТ

Отчет по лабораторной работе №5

по дисциплине

«Операционные системы и системное программирование»

Выполнил:

Гладкий М.Г.

группа 851005

Проверил:

Деменковец Д.В.

Минск, 2020

**Исходный код программы**

Main.cpp

#include <iostream>  
#include <queue>  
#include <vector>  
#include <string>  
#include <fstream>  
#include <thread>  
#include "queueElem/QueueElem.h"  
#include "queueHandler/ThreadpoolQueueProcessor.h"  
  
#define MAX\_STRING\_SIZE 1024  
  
using namespace std;  
  
int main() {  
 Queue myQueue;  
 vector<string> buffer;  
 int countThreadsConsole;  
 unsigned int maxThreads = thread::hardware\_concurrency();  
 bool isRight;  
 isRight = false;  
 do {  
 cout << "Number of threads to use:" << endl;  
 cin >> countThreadsConsole;  
 if (countThreadsConsole > maxThreads){  
 cout << "Max number of threads is " << maxThreads << ". Try again."<< endl;  
 } else {  
 isRight = true;  
 }  
 } while (!isRight);  
  
 ifstream streamIn("C:\\Users\\maksf\\Desktop\\OSISP\\Lab5\\input.txt");  
 long strCount = 0;  
 char str[MAX\_STRING\_SIZE];  
 int j = 0;  
 while (!streamIn.eof()){  
 streamIn.getline(str,MAX\_STRING\_SIZE);  
 strCount++;  
 buffer.resize(strCount);  
 buffer[j] = str;  
 if (buffer[j][buffer[j].size() - 1] != '\n'){  
 buffer[j] += '\n';  
 }  
 j++;  
 }  
 streamIn.close();  
 cout << "File has been read." << endl;  
  
 int countThreads;  
 long strCountForThread;  
 int countMod;  
 if (strCount > countThreadsConsole) {  
 countThreads = countThreadsConsole;  
 strCountForThread = strCount / countThreadsConsole;  
 countMod = strCount % countThreadsConsole;  
 } else {  
 strCountForThread = 1;  
 countMod = 0;  
 countThreads = strCount;  
 }  
 for (int i = 0; i < countThreads; i++) {  
 if (i != countThreads - 1) {  
 QueueElem queueElem(strCountForThread \* i + 1, strCountForThread \* (i + 1));  
 myQueue.pushInQueue(queueElem);  
 } else {  
 QueueElem queueElem(strCountForThread \* i + 1, strCountForThread \* (i + 1) + countMod);  
 myQueue.pushInQueue(queueElem);  
 }  
 }  
  
 ThreadpoolQueueProcessor queueHandle(&myQueue,&buffer);  
 queueHandle.Process(countThreads);  
 queueHandle.Wait();  
  
 int countSort = strCount / strCountForThread - 1;  
 for (int i = 0; i < countSort; i++){  
 if (i == (countSort - 1)) {  
 inplace\_merge(buffer.begin(), buffer.begin() + strCountForThread \* (i + 1),  
 buffer.end());  
 } else {  
 inplace\_merge(buffer.begin(), buffer.begin() + strCountForThread \* (i + 1),  
 buffer.begin() + strCountForThread \* (i + 2));  
 }  
 }  
  
 ofstream streamOut("C:\\Users\\maksf\\Desktop\\OSISP\\Lab5\\output.txt");  
 for(int i = 0; i < buffer.size(); i++){  
 streamOut << buffer[i];  
 }  
 streamOut.close();  
  
 cout << "Answer has been written." << endl;  
 cout << "Done!" << endl;  
  
 return 0;  
}

Queue.cpp

#include "Queue.h"  
  
void Queue::pushInQueue(QueueElem queueElement){  
 EnterCriticalSection(&pushCriticalSection);  
 this->myQueue.push(queueElement);  
 LeaveCriticalSection(&pushCriticalSection);  
}  
  
QueueElem Queue::popFromQueue(){  
 EnterCriticalSection(&popCriticalSection);  
 QueueElem result = this->myQueue.front();  
 this->myQueue.pop();  
 LeaveCriticalSection(&popCriticalSection);  
 return result;  
}  
  
Queue::Queue() {  
 InitializeCriticalSection(&popCriticalSection);  
 InitializeCriticalSection(&pushCriticalSection);  
}  
  
Queue::~Queue() {  
 DeleteCriticalSection(&popCriticalSection);  
 DeleteCriticalSection(&pushCriticalSection);  
}

Queue.h

#ifndef LAB5\_QUEUE\_H  
#define LAB5\_QUEUE\_H  
  
#include <queue>  
#include <windows.h>  
#include "../queueElem/QueueElem.h"  
  
using namespace std;  
  
class Queue {  
public:  
 queue<QueueElem> myQueue;  
 CRITICAL\_SECTION pushCriticalSection;  
 CRITICAL\_SECTION popCriticalSection;  
 Queue();  
 ~Queue();  
 void pushInQueue(QueueElem Element);  
 QueueElem popFromQueue();  
};  
  
#endif

QueueElem.cpp

#include "QueueElem.h"  
  
 QueueElem::QueueElem(long start, long finish) {  
 this->startOffset = start;  
 this->finishOffset = finish;  
}

QueueElem.h

#ifndef LAB5\_QUEUEELEM\_H  
#define LAB5\_QUEUEELEM\_H  
  
class QueueElem {  
public:  
 long startOffset{ 0 };  
 long finishOffset{ 0 };  
 QueueElem(long start, long finish);  
};  
  
#endif

ThreadpoolQueueProcessor.cpp

#include "ThreadpoolQueueProcessor.h"  
#include <Windows.h>  
#include <iostream>  
#include <queue>  
  
ThreadpoolQueueProcessor::ThreadpoolQueueProcessor(Queue \*MyQueue, vector<string> \*Buffer) {  
 InitializeThreadpoolEnvironment(&this->callbackEnvironment);  
 this->pool = CreateThreadpool(nullptr);  
 SetThreadpoolThreadMinimum(this->pool, 1);  
 this->cleanupGroup = CreateThreadpoolCleanupGroup();  
 SetThreadpoolCallbackCleanupGroup(&this->callbackEnvironment, this->cleanupGroup, nullptr);  
 myQueue = MyQueue;  
 buffer = Buffer;  
}  
  
void ThreadpoolQueueProcessor::SortFile(QueueElem queueElem){  
 int startIndex = queueElem.startOffset - 1;  
 int finishIndex = queueElem.finishOffset;  
 sort(buffer->begin()+startIndex,buffer->begin() + finishIndex);  
}  
  
void ThreadpoolQueueProcessor::queueElementHandler(ThreadpoolQueueProcessor \*queueHandler){  
 QueueElem queueElement = queueHandler->myQueue->popFromQueue();  
 queueHandler->SortFile(queueElement);  
}  
  
ThreadpoolQueueProcessor::~ThreadpoolQueueProcessor() {  
 CloseThreadpoolCleanupGroup(this->cleanupGroup);  
 CloseThreadpool(this->pool);  
}  
  
void ThreadpoolQueueProcessor::Process(int threadCount) {  
 SetThreadpoolThreadMaximum(this->pool, threadCount);  
 PTP\_WORK work = CreateThreadpoolWork(WorkCallback, this, &this->callbackEnvironment);  
 for (int i = 0; i < threadCount; i++) {  
 SubmitThreadpoolWork(work);  
 }  
}  
  
void ThreadpoolQueueProcessor::Wait() {  
 CloseThreadpoolCleanupGroupMembers(this->cleanupGroup, false, nullptr);  
}  
  
void ThreadpoolQueueProcessor::WorkCallback(PTP\_CALLBACK\_INSTANCE instance, PVOID parameter, PTP\_WORK work) {  
 auto \*myThis = reinterpret\_cast<ThreadpoolQueueProcessor \*>(parameter);  
 queueElementHandler(myThis);  
}

ThreadpoolQueueProcessor.h

#ifndef LAB5\_THREADPOOLQUEUEPROCESSOR\_H  
#define LAB5\_THREADPOOLQUEUEPROCESSOR\_H  
  
#include <Windows.h>  
#include <queue>  
#include <string>  
  
#include "../queueElem/QueueElem.h"  
#include "../queue/Queue.h"  
  
using namespace std;  
  
class ThreadpoolQueueProcessor {  
public:  
 Queue \*myQueue;  
 vector<string> \*buffer;  
 ThreadpoolQueueProcessor(Queue \*MyQueue, vector<string>\* Buffer);  
 ~ThreadpoolQueueProcessor();  
 void Process(int threadCount);  
 void Wait();  
  
private:  
 PTP\_POOL pool;  
 PTP\_CLEANUP\_GROUP cleanupGroup;  
 TP\_CALLBACK\_ENVIRON callbackEnvironment{};  
 void SortFile(QueueElem queueElem);  
 static void queueElementHandler(ThreadpoolQueueProcessor \*queueHandler);  
 static void CALLBACK WorkCallback(PTP\_CALLBACK\_INSTANCE instance, PVOID parameter, PTP\_WORK work);  
};  
  
#endif

**Скриншоты выполнения программы**





