Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

Студент: Рамалданов Рустамхан Ражудинович

Группа: М8О-208Б-22

Вариант: 3

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2023

**Содержание**

1. Репозиторий

2. Постановка задачи

3. Общие сведения о программе

4. Общий метод и алгоритм решения

5. Исходный код

6. Демонстрация работы программы

7. Выводы

**Репозиторий**

<https://github.com/pepelulka/OS_labs>

**Постановка задачи**

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). Ограничение максимального количества потоков, работающих в один момент времени, должно быть задано ключом запуска вашей программы.

Так же необходимо уметь продемонстрировать количество потоков,

используемое вашей программой с помощью стандартных средств

операционной системы.

В отчете привести исследование зависимости ускорения и эффективности

алгоритма от входных данных и количества потоков. Получившиеся

результаты необходимо объяснить.

**Общие сведения о программе**

Программа написана под операционную систему Windows и, соответственно, использует библиотеку системных вызовов “windows.h”.

Программе в аргументах командной строки передаются имя файла с числами для сортировки и количество потоков, на которые будет распараллелен алгоритм. Программа производит расчеты времени, затраченного на сортировку в случае одного потока и нескольких.

**Общий метод и алгоритм решения**

Вся многопоточность реализуется в классе TThreadPool, который предоставляет интерфейс очереди задач, в которую в любой момент можно положить очередную задачу, а потом, когда в нем наберется необходимое количество задач, все они начнут выполняться параллельно на предварительно запущенных потоках. Сама сортировка проходит в несколько этапов, на первом этапе сортируются все куски массива по 2, на втором этапе – все куски по 4, путем слияния полученных на предыдущем этапе кусков, и так пока не будет отсортирован весь массив.

**Исходный код**

**========================== generator.h ==========================**

**#pragma once**

**#include <algorithm>**

**#include <fstream>**

**#include <vector>**

**#include <random>**

**namespace gen {**

**void WriteToFile(const std::vector<int> &vec, const std::string &filename);**

**std::vector<int> GenerateSorted(size\_t n);**

**std::vector<int> GenerateReversed(size\_t n);**

**std::vector<int> GenerateRandomShuffle(size\_t n);**

**}**

**========================== merge\_sort.h ==========================**

**#pragma once**

**#include <iostream>**

**#include <vector>**

**#include "thread\_pool.h"**

**namespace lab2 {**

**void MergeSort(std::vector<int> &vec);**

**void ParallelMergeSort(std::vector<int> &vec, size\_t threadCount);**

**}**

**========================== thread\_pool.h ==========================**

**#pragma once**

**#ifdef \_WIN32**

**//========================//**

**// Windows //**

**//========================//**

**#include <iostream>**

**#include <queue>**

**#include <functional>**

**#include <memory>**

**#include <cassert>**

**#include "windows.h"**

**#include "process.h"**

**namespace tp {**

**class TThreadPool final {**

**public:**

**using TaskFunction = std::function<VOID(PVOID)>;**

**struct TTask {**

**TaskFunction func;**

**std::shared\_ptr<VOID> data;**

**};**

**TThreadPool() = delete;**

**TThreadPool(WORD threadNum);**

**~TThreadPool();**

**VOID PushTask(WORD thread, TTask task);**

**VOID Execute();**

**VOID Terminate();**

**private:**

**struct TThread {**

**HANDLE thread;**

**TThreadPool \*owner;**

**// Queue**

**CRITICAL\_SECTION csQueue;**

**CONDITION\_VARIABLE cvQueue;**

**std::queue<TTask> queue;**

**static unsigned WINAPI ThreadRoutine(PVOID data);**

**VOID Start();**

**TThread() {**

**InitializeCriticalSection(&csQueue);**

**InitializeConditionVariable(&cvQueue);**

**}**

**~TThread() {**

**DeleteCriticalSection(&csQueue);**

**}**

**};**

**struct TCompletenessCounter {**

**DWORD count, target;**

**CRITICAL\_SECTION cs;**

**CONDITION\_VARIABLE cv;**

**TCompletenessCounter() {**

**InitializeCriticalSection(&cs);**

**InitializeConditionVariable(&cv);**

**count = 0;**

**}**

**~TCompletenessCounter() {**

**DeleteCriticalSection(&cs);**

**}**

**VOID WaitTarget() {**

**EnterCriticalSection(&cs);**

**if (count != target) {**

**SleepConditionVariableCS(&cv, &cs, INFINITE);**

**}**

**LeaveCriticalSection(&cs);**

**}**

**VOID ResetAndSetTarget(DWORD target) {**

**EnterCriticalSection(&cs);**

**count = 0;**

**this->target = target;**

**LeaveCriticalSection(&cs);**

**}**

**VOID Inc() {**

**EnterCriticalSection(&cs);**

**count++;**

**if (count == target) {**

**WakeAllConditionVariable(&cv);**

**}**

**LeaveCriticalSection(&cs);**

**}**

**};**

**WORD threadNum;**

**BOOL isTerminated;**

**std::vector<TThread> threads;**

**TCompletenessCounter counter;**

**};**

**}**

**#endif**

**========================== timer.h ==========================**

**#pragma once**

**#include <chrono>**

**#include <iostream>**

**namespace lab2 {**

**class Timer {**

**public:**

**using Clock = std::chrono::high\_resolution\_clock;**

**using Unit = std::chrono::milliseconds;**

**Timer();**

**void reset();**

**int get();**

**private:**

**std::chrono::time\_point<Clock> timePoint;**

**};**

**} // namespace lab2**

**========================== generator.cpp ==========================**

**#include "generator.h"**

**namespace gen {**

**void WriteToFile(const std::vector<int> &vec, const std::string &filename) {**

**size\_t size = vec.size();**

**std::ofstream out(filename);**

**out << size << std::endl;**

**for (const auto &i : vec) {**

**out << i << " ";**

**}**

**}**

**std::vector<int> GenerateSorted(size\_t n) {**

**std::vector<int> result;**

**for (size\_t i = 0; i < n; i++) {**

**result.push\_back(i);**

**}**

**return result;**

**}**

**std::vector<int> GenerateReversed(size\_t n) {**

**std::vector<int> result;**

**for (size\_t i = 0; i < n; i++) {**

**result.push\_back(n - i - 1);**

**}**

**return result;**

**}**

**std::vector<int> GenerateRandomShuffle(size\_t n) {**

**std::vector<int> result;**

**for (size\_t i = 0; i < n; i++) {**

**result.push\_back(i);**

**}**

**auto rd = std::random\_device {};**

**auto rng = std::default\_random\_engine { rd() };**

**std::shuffle(std::begin(result), std::end(result), rng);**

**return result;**

**}**

**}**

**========================== merge\_sort.cpp ==========================**

**#include "merge\_sort.h"**

**namespace lab2 {**

**using TSegment = std::pair<size\_t, size\_t>;**

**static void Merge(std::vector<int> &vec, TSegment lseg, TSegment rseg,**

**std::vector<int> &extra, size\_t extraArrayStart) {**

**size\_t l1 = lseg.first, r1 = lseg.second, l2 = rseg.first, r2 = rseg.second;**

**size\_t i = l1, j = l2;**

**size\_t extraPtr = extraArrayStart;**

**while (i <= r1 || j <= r2) {**

**if (i > r1) {**

**extra[extraPtr] = vec[j];**

**j++;**

**} else if (j > r2) {**

**extra[extraPtr] = vec[i];**

**i++;**

**} else {**

**if (vec[i] < vec[j]) {**

**extra[extraPtr] = vec[i];**

**i++;**

**} else {**

**extra[extraPtr] = vec[j];**

**j++;**

**}**

**}**

**extraPtr++;**

**}**

**for (size\_t i = extraArrayStart; i < extraPtr; i++) {**

**vec[l1 + i - extraArrayStart] = extra[i];**

**}**

**}**

**void MergeSort(std::vector<int> &vec) {**

**size\_t size = vec.size();**

**std::vector<int> extra(size);**

**for (size\_t k = 1; k < size; k \*= 2) {**

**// Segments of the array to merge**

**for (int i = 0; i + k < size; i += 2 \* k) {**

**size\_t l1, r1, l2, r2;**

**l1 = i;**

**r1 = i + k - 1;**

**l2 = i + k;**

**if (i + 2 \* k - 1 < size - 1) {**

**r2 = i + 2 \* k - 1;**

**} else {**

**r2 = size - 1;**

**}**

**Merge(vec, {l1, r1}, {l2, r2}, extra, 0);**

**}**

**}**

**}**

**// ==================//**

**// Parallel sort**

**// ==================//**

**struct TMergeRawInputData {**

**std::vector<int> \*vec, \*extra;**

**TSegment lseg, rseg;**

**size\_t extraArrayStart;**

**TMergeRawInputData(std::vector<int> \*vec1, std::vector<int> \*extra1,**

**TSegment lseg1, TSegment rseg1, size\_t extraArrayStart1) :**

**vec(vec1), extra(extra1), lseg(lseg1), rseg(rseg1), extraArrayStart(extraArrayStart1) { }**

**};**

**static void\* MergeRaw(void \*data) {**

**TMergeRawInputData \*input = static\_cast<TMergeRawInputData\*>(data);**

**Merge(\*input->vec, input->lseg, input->rseg, \*input->extra, input->extraArrayStart);**

**return nullptr;**

**}**

**void ParallelMergeSort(std::vector<int> &vec, size\_t threadCount) {**

**using TTask = tp::TThreadPool::TTask;**

**size\_t size = vec.size();**

**std::vector<int> extra(size);**

**tp::TThreadPool tpool(threadCount);**

**for (size\_t k = 1; k < size; k \*= 2) {**

**// %%% <%OPT>**

**if (k <= 4) {**

**for (int i = 0; i + k < size; i += 2 \* k) {**

**size\_t l1, r1, l2, r2;**

**l1 = i;**

**r1 = i + k - 1;**

**l2 = i + k;**

**if (i + 2 \* k - 1 < size - 1) {**

**r2 = i + 2 \* k - 1;**

**} else {**

**r2 = size - 1;**

**}**

**Merge(vec, {l1, r1}, {l2, r2}, extra, 0);**

**}**

**continue;**

**}**

**// %% <%/OPT>**

**// Segments of the array to merge**

**int ptr = 0;**

**for (int i = 0; i + k < size; i += 2 \* k) {**

**size\_t l1, r1, l2, r2;**

**size\_t extraStart = i;**

**l1 = i;**

**r1 = i + k - 1;**

**l2 = i + k;**

**if (i + 2 \* k - 1 < size - 1) {**

**r2 = i + 2 \* k - 1;**

**} else {**

**r2 = size - 1;**

**}**

**auto data = std::make\_shared<TMergeRawInputData>(TMergeRawInputData(&vec,**

**&extra,**

**{l1, r1},**

**{l2, r2},**

**extraStart));**

**tpool.PushTask(ptr % threadCount, TTask{&MergeRaw, data});**

**ptr++;**

**}**

**tpool.Execute();**

**}**

**tpool.Terminate();**

**}**

**}**

**========================== thread\_pool.cpp ==========================**

**#ifdef \_WIN32**

**#include "thread\_pool.h"**

**namespace tp {**

**TThreadPool::TThreadPool(WORD threadNum) {**

**isTerminated = false;**

**this->counter.ResetAndSetTarget(threadNum);**

**this->threadNum = threadNum;**

**// Dangerous!! May be reallocation inside vector!**

**this->threads.resize(threadNum);**

**for (WORD i = 0; i < threadNum; i++) {**

**this->threads[i].owner = this;**

**this->threads[i].Start();**

**}**

**this->counter.WaitTarget();**

**this->counter.ResetAndSetTarget(threadNum);**

**}**

**TThreadPool::~TThreadPool() { }**

**VOID TThreadPool::PushTask(WORD thread, TTask task) {**

**assert(thread < threadNum);**

**threads[thread].queue.push(task);**

**}**

**VOID TThreadPool::Execute() {**

**for (auto &th : threads) {**

**EnterCriticalSection(&th.csQueue);**

**WakeConditionVariable(&th.cvQueue);**

**LeaveCriticalSection(&th.csQueue);**

**}**

**this->counter.WaitTarget();**

**this->counter.ResetAndSetTarget(threadNum);**

**}**

**VOID TThreadPool::Terminate() {**

**isTerminated = true;**

**Execute();**

**for (auto &th : threads) {**

**WaitForSingleObject(th.thread, INFINITE);**

**CloseHandle(th.thread);**

**}**

**}**

**unsigned WINAPI TThreadPool::TThread::ThreadRoutine(PVOID data) {**

**TThread \*thread = (TThread\*)data;**

**while (true) {**

**EnterCriticalSection(&thread->csQueue);**

**while (!thread->queue.empty()) {**

**TTask task = thread->queue.front();**

**thread->queue.pop();**

**task.func(task.data.get());**

**}**

**if (thread->owner->isTerminated) {**

**LeaveCriticalSection(&thread->csQueue);**

**thread->owner->counter.Inc();**

**return 0;**

**}**

**thread->owner->counter.Inc();**

**SleepConditionVariableCS(&thread->cvQueue, &thread->csQueue, INFINITE);**

**LeaveCriticalSection(&thread->csQueue);**

**}**

**return 0;**

**}**

**VOID TThreadPool::TThread::Start() {**

**thread = (HANDLE)\_beginthreadex(0, 0, &ThreadRoutine, this, 0, 0);**

**}**

**}**

**#endif**

**========================== timer.cpp ==========================**

**#include "timer.h"**

**namespace lab2 {**

**Timer::Timer() {**

**this->reset();**

**}**

**void Timer::reset() {**

**this->timePoint = Clock::now();**

**}**

**int Timer::get() {**

**auto timeNow = Clock::now();**

**return std::chrono::duration\_cast<Unit>(timeNow - this->timePoint).count();**

**}**

**} // namespace lab2**

**========================== main.cpp ==========================**

**#include "timer.h"**

**#include "thread\_pool.h"**

**#include "merge\_sort.h"**

**#include <fstream>**

**#include <cassert>**

**using namespace lab2;**

**std::vector<int> ReadFromFile(const std::string &fileName) {**

**std::ifstream in(fileName);**

**std::vector<int> result;**

**size\_t n;**

**in >> n;**

**result.resize(n);**

**for (size\_t i = 0; i < n; i++) {**

**in >> result[i];**

**}**

**return result;**

**}**

**// Usage:**

**// <executable> <filename> <thread-count>**

**int main(int argc, char \*\*argv) {**

**int threadNum = 1;**

**std::string filename = "vec.txt";**

**if (argc >= 2) {**

**filename = std::string(argv[1]);**

**}**

**if (argc >= 3) {**

**threadNum = std::atoi(argv[2]);**

**}**

**Timer timer;**

**std::vector<int> a = ReadFromFile(filename);**

**std::vector<int> b = a;**

**std::cout << "Read vector from file " << filename << std::endl;**

**timer.reset();**

**MergeSort(a);**

**std::cout << "time1: " << timer.get() << "ms\n";**

**timer.reset();**

**ParallelMergeSort(b, threadNum);**

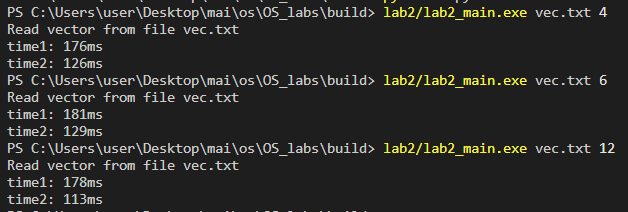
**std::cout << "time2: " << timer.get() << "ms\n";**

**assert(std::is\_sorted(a.begin(), a.end()));**

**assert(std::is\_sorted(b.begin(), b.end()));**

**}**

**Демонстрация работы программы**



**Выводы**

В ходе выполнения лабораторной работе я получил знания и навыки использования системных вызовов Windows при написании программ. Я узнал о различиях устройства операционных систем Linux и Windows и о нюансах работы с системными вызовами Windows. Было выяснено, что Windows, в отличие от Linux, не имеет открытой документации системных вызовов.