Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЁТ

к лабораторной работе №1

на тему

**УПРАВЛЕНИЕ ПРОЦЕССАМИ, ПОТОКАМИ, НИТЯМИ**

Выполнил: студент гр.253504 Новиков В.А.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc179907483)

[2 Краткие теоретические сведения 4](#_Toc179907484)

[3 Описание функций и работы программы 5](#_Toc179907485)

[3.1 Описание функций программы 5](#_Toc179907486)

[3.2 Работа программы 5](#_Toc179907487)

[Заключение 7](#_Toc179907488)

[Список использованных источников 8](#_Toc179907489)

Приложение А[(обязательное) 9](#_Toc179907491)

# **1 ЦЕЛЬ РАБОТЫ**

Целью данной работы является из возобновление, закрепление и развитие навыков программирования приложений Windows; изучение концепции вычислительных процессов, потоков и нитей и их реализации в Windows; изучение основных этапов жизненного цикла процессов и потоков, элементарного управление ими: порождение, завершение, получение и изменение состояния. Типичное (простое) использование многозадачности

В данной лабораторной работе требуется разработать программу для многопоточной обработки данных. Данные представляют собой две матрицы, перемножение которых производится при помощи потоков с записью результата в матрицу результата.

Программным продуктом лабораторной работы является консольное приложение, написанное на языке программирования C++ с использованием API операционной системы Windows. Основные требования к программе:

– массив данных разбивается на несколько частей(фрагментов);

– каждый фрагмент обрабатывается отдельным потоком;

– количество потоков и размер массива данных задаются пользователем;

– отображение прогресса выполнения;

– отображение времени выполнения потока.

Для разработки программного продукта использовалась среда разработки Microsoft Visual Studio 2022, предоставляющая инструменты для написания, компиляции, отладки и тестирования приложений на C++. В Visual Studio был реализован весь цикл разработки — от написания исходного кода до отладки программы и её запуска на тестовых данных. Использование Visual Studio позволило автоматизировать процесс компиляции и управления проектом, а также отследить ошибки на этапе разработки.

В отчёте по лабораторной работе также были описаны ключевые понятия, касающиеся процессов, потоков и нитей.

2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Приложение состоит из одного или нескольких процессов. Процесс, в простейших терминах, — это выполняющаяся программа. Один или несколько потоков выполняются в контексте процесса.

Каждый процесс предоставляет ресурсы, необходимые для выполнения программы. Процесс имеет виртуальное адресное пространство, исполняемый код, открытые дескрипторы для системных объектов, контекст безопасности, уникальный идентификатор процесса, переменные среды, класс приоритета, минимальный и максимальный размер рабочего набора и по крайней мере один поток выполнения. Каждый процесс запускается с одного потока, часто называемого основным потоком, но может создавать дополнительные потоки из любого из его потоков.

Поток — это сущность в процессе, которую можно запланировать для выполнения. Все потоки процесса совместно используют его виртуальное адресное пространство и системные ресурсы. Кроме того, каждый поток поддерживает обработчики исключений, приоритет планирования, локальное хранилище потока, уникальный идентификатор потока и набор структур, которые система будет использовать для сохранения контекста потока до его планирования. Контекст потока включает набор регистров компьютера потока, стек ядра, блок среды потока и стек пользователя в адресном пространстве процесса потока. Потоки также могут иметь собственный контекст безопасности, который можно использовать для олицетворения клиентов.

Microsoft Windows поддерживает упреждающую многозадачность, что создает эффект одновременного выполнения нескольких потоков из нескольких процессов. На многопроцессорном компьютере система может одновременно выполнять столько потоков, сколько процессоров на компьютере.

Объект задания позволяет управлять группами процессов как единое целое. Объекты заданий — это именуемые, защищаемые, общие объекты, управляющие атрибутами связанных с ними процессов. Операции, выполняемые с объектом задания, влияют на все процессы, связанные с объектом задания.

Приложение может использовать пул потоков, чтобы уменьшить количество потоков приложения и обеспечить управление рабочими потоками. Приложения могут ставить рабочие элементы в очередь, связывать работу с дескрипторами ожидания, автоматически ставить в очередь по таймеру и выполнять привязку с помощью операций ввода-вывода.[1]

3 ОПИСАНИЕ ФУНКЦИЙ И РАБОТА ПРОГРАММЫ

## **3.1 Описание функций программы**

Программа представляет собой консольное приложение, в котором согласно формулировке задачи были спроектированы следующие функции:

– массив данных разбивается на несколько частей(фрагментов);

– каждый фрагмент обрабатывается отдельным потоком;

– количество потоков и размер массива данных задаются пользователем;

– отображение прогресса выполнения;

– отображение времени выполнения потока.

Выбор размера матрицы и количества потоков реализован при помощи консольного ввода. Создание и взаимодействие с потоками происходит с помощью функций Windows API, таких как CreateThread и WaitForMultipleObjects.[2] Подсчет времени реализован при помощи функций QueryPerformanceFrequency и QueryPerformanceCounter.[3] Отображение прогресса выполнения потока происходит при помощи функций перемещения курсора, а также критических секций с использованием функций EnterCriticalSection, DeleteCriticalSection.[4]

## **3.2 Работа программы**

При запуске программы пользователю необходимо выбрать размер перемножаемых матриц и количество потоков для произведения вычислений, что показано на рисунке 3.1.

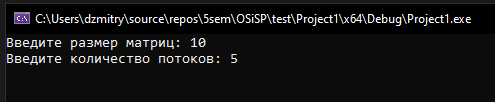


Рисунок 3.1 – Ввод данных пользователем

После вводом пользователя размера матриц и количества потоков создается три матрицы:

– матрица А, заполненная значением 1;

– матрица B, заполненная значением 2;

– матрица С для записи результата.

Далее матрицы делятся на фрагменты, обработку которых будут производить потоки, после чего создаются сами потоки в количестве, выбранным пользователем, и производят вычисление выделенных им фрагментов. Пользователю выводится информация о прогрессе выполнения потока. Процесс выполнения программы изображен на рисунке 3.2.

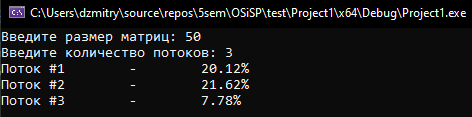


Рисунок 3.2 – Выполнение программы

По окончанию выполнения потока выводится время, за которое он выполнился, а по завершении выполнения всех потоков выводится результат перемножения матриц, что изображено на рисунке 3.3.

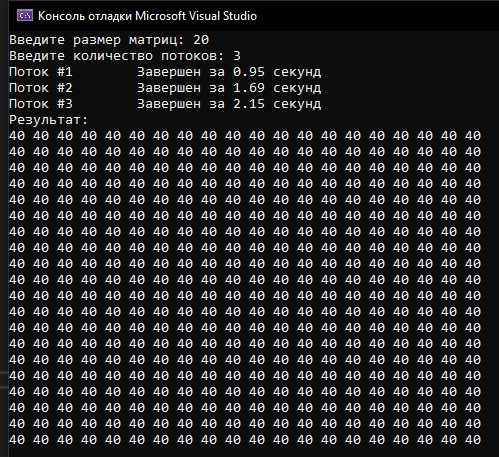


Рисунок 3.3 – Результат выполнения программы

ЗАКЛЮЧЕНИЕ

В ходе данной работы был разработан программный продукт, который выполняет многопоточную обработку данных, в частности перемножение матриц. Этот программный продукт стал инструментом для изучения влияния различных факторов на производительность многопоточной обработки данных.

Реализация программы включала в себя использование функций WinAPI для управления потоками, таких как CreateThread для создания новых потоков, WaitForMultipleObjects для ожидания завершения нескольких потоков и CloseHandle для закрытия дескрипторов потоков. Эти функции являются основными инструментами для работы с потоками в Windows, обеспечивая возможность прямого взаимодействия с операционной системой на уровне управления потоками.

Программа успешно продемонстрировала возможность создания и управления потоками в соответствии с заданными условиями.

Лабораторная работа позволила закрепить знания о работе с процессами, потоками и функциями WinAPI, а также о важности управления ресурсами операционной системы для обеспечения корректной работы приложений и избегания утечек памяти и некорректных состояний процессов, потоков и нитей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Build desktop Windows apps using the Win32 API [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/

[2] Процессы и потоки (Win32 API). Программирование в ОС Windows. Лекция 4. [Электронный ресурс]. – Режим доступа: https://youtu.be/c\_\_SQwCOcBc?si=EtOcywJ2F-fmIUsQ

[3] 30. Query Performance Counter - Windows System Programming in C/C++. [Электронный ресурс]. – Режим доступа: https://youtu.be/9YAtPpgcSWE?si=DRGnQkS\_XDY5X-Hp

[4] 21.Critical Section for Thread Synchronization - Windows System Programming in C/C++. [Электронный ресурс]. – Режим доступа: https://youtu.be/jNnMF7r6SkU?si=6urjs\_FH4xDDZ4oB

ПРИЛОЖЕНИЕ А

(обязательное)

**Исходный код программы**

#include <windows.h>

#include <iostream>

#include <vector>

#include <string>

#include <fstream>

#include <iomanip>

class ThreadData {

public:

int threadId;

int size;

long long startRow;

long long rowCount;

double percent;

COORD cursorPos;

std::vector<std::vector<int>>& A;

std::vector<std::vector<int>>& B;

std::vector<std::vector<int>>& C;

ThreadData(int id, long long start, long long count, COORD pos,

std::vector<std::vector<int>>& a,

std::vector<std::vector<int>>& b,

std::vector<std::vector<int>>& c,

int msize)

: threadId(id), startRow(start), rowCount(count), cursorPos(pos), A(a), B(b), C(c), percent(0), size(msize) {}

};

CRITICAL\_SECTION consoleLock;

void MoveCursorToPosition(int x, int y) {

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

COORD position;

position.X = x;

position.Y = y;

SetConsoleCursorPosition(hConsole, position);

}

void HideCursor() {

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

CONSOLE\_CURSOR\_INFO cursorInfo;

GetConsoleCursorInfo(hConsole, &cursorInfo);

cursorInfo.bVisible = FALSE;

SetConsoleCursorInfo(hConsole, &cursorInfo);

}

void ShowCursor() {

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

CONSOLE\_CURSOR\_INFO cursorInfo;

GetConsoleCursorInfo(hConsole, &cursorInfo);

cursorInfo.bVisible = TRUE;

SetConsoleCursorInfo(hConsole, &cursorInfo);

}

DWORD WINAPI ProcessMatrixSegment(LPVOID lpParam) {

ThreadData\* data = static\_cast<ThreadData\*>(lpParam);

LARGE\_INTEGER frequency, start, end;

QueryPerformanceFrequency(&frequency);

QueryPerformanceCounter(&start);

long long elementsCount = data->rowCount \* data->size;

long long counter = 0;

for (long long i = data->startRow; i < data->rowCount + data->startRow; ++i) {

data->percent = (i + 1) \* 100.0 / data->rowCount;

for (long long j = 0; j < data->size; ++j) {

data->C[i][j] = 0;

for (long long k = 0; k < data->size; ++k) {

data->C[i][j] += data->A[i][k] \* data->B[k][j];

}

++counter;

double up = ((i - data->startRow) \* data->size + j);

double down = data->rowCount \* data->size;

data->percent = ((double)counter / (double)elementsCount) \* 100.0;

EnterCriticalSection(&consoleLock);

MoveCursorToPosition(data->cursorPos.X, data->cursorPos.Y);

std::cout << std::fixed << std::setprecision(2)

<< "Поток #" << data->threadId << "\t-\t " << data->percent << "% ";

LeaveCriticalSection(&consoleLock);

//Sleep(100);

}

}

QueryPerformanceCounter(&end);

double elapsedTime = static\_cast<double>(end.QuadPart - start.QuadPart) / frequency.QuadPart;

EnterCriticalSection(&consoleLock);

MoveCursorToPosition(data->cursorPos.X, data->cursorPos.Y);

std::cout << "\t\t\t";

MoveCursorToPosition(data->cursorPos.X, data->cursorPos.Y);

std::cout << std::fixed << std::setprecision(2)

<< "Поток #" << data->threadId << "\tЗавершен за " << elapsedTime << " секунд ";

LeaveCriticalSection(&consoleLock);

return 0;

}

int main() {

setlocale(0, "");

InitializeCriticalSection(&consoleLock);

int threadCount;

int MATRIX\_SIZE;

std::cout << "Введите размер матриц: ";

std::cin >> MATRIX\_SIZE;

if (MATRIX\_SIZE <= 1) return 1;

std::vector<std::vector<int>> A(MATRIX\_SIZE, std::vector<int>(MATRIX\_SIZE, 1)); // Матрица A

std::vector<std::vector<int>> B(MATRIX\_SIZE, std::vector<int>(MATRIX\_SIZE, 2)); // Матрица B

std::vector<std::vector<int>> C(MATRIX\_SIZE, std::vector<int>(MATRIX\_SIZE, 0)); // Матрица результата C

std::cout << "Введите количество потоков: ";

std::cin >> threadCount;

if (threadCount <= 0 || threadCount > MATRIX\_SIZE) return 1;

CONSOLE\_SCREEN\_BUFFER\_INFO csbi;

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

GetConsoleScreenBufferInfo(hConsole, &csbi);

COORD originalPosition = csbi.dwCursorPosition;

COORD pos = { 0, 0 };

long long rowCount = MATRIX\_SIZE / threadCount;

long long lastRowCount = rowCount + MATRIX\_SIZE % threadCount;

HANDLE\* threads = new HANDLE[threadCount];

std::vector<ThreadData\*> threadData;

int i = 0;

for (i = 0; i < threadCount - 1; i++)

{

pos.X = originalPosition.X;

pos.Y = originalPosition.Y + i;

threadData.push\_back(new ThreadData(i + 1, rowCount \* i, rowCount, pos, A, B, C, MATRIX\_SIZE));

}

pos.X = originalPosition.X;

pos.Y = originalPosition.Y + i;

threadData.push\_back(new ThreadData(i + 1, rowCount \* i, lastRowCount, pos, A, B, C, MATRIX\_SIZE));

HideCursor();

for (int i = 0; i < threadCount; ++i) {

threads[i] = CreateThread(NULL, 0, ProcessMatrixSegment, threadData[i], 0, NULL);

if (threads[i] == NULL) {

std::cerr << "Ошибка при создании потока " << i + 1 << std::endl;

return 1;

}

}

WaitForMultipleObjects(threadCount, threads, TRUE, INFINITE);

for (int i = 0; i < threadCount; ++i) {

CloseHandle(threads[i]);

}

MoveCursorToPosition(0, threadCount + 2);

std::cout << "Результат:\n";

for (long long i = 0; i < MATRIX\_SIZE; ++i) {

for (long long j = 0; j < MATRIX\_SIZE; ++j) {

std::cout << C[i][j] << " ";

}

std::cout << std::endl;

}

delete[] threads;

DeleteCriticalSection(&consoleLock);

ShowCursor();

return 0;

}