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 Лавренова А.С.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

Введение5

[1 Ц](#_agswffz6g6ei)ель работы6

[2](#_8c2l0lszpix4) Теоретические сведения11

[2.1 Управление процессами](#_g9qyph9usapd) 11

[2.2 Истроия, версии и достоинства](#_mguzp67fty5w) 12

[2.3 Обоснование выбора платформы](#_yh7snu680n92) 13

[2.4 Анализ операционной системы для написания программы](#_yh7snu680n92) 13

[3 Теоретическое обоснование разработки программного продукта](#_1mzm3wfnooze) 14

[3.1 Обоснование необходимости разработки](#_ifb4a7rfcyp7) 15

[3.2 Технологии программирования, используемые для решения поставленных задач](#_iglf44ipldgx) 22

[3.3 Связь архитектуры вычислительной системы с разрабатываемым программным обеспечением](#_ens82q1l1dp6) 23

[4](#_8c2l0lszpix4) Проектирование функциональных возможностей программы11

[4.1 Обоснования и описание функций программного обеспечения](#_g9qyph9usapd) 11

Заключение28

Список использованных источников29

Приложение [А](#_nypodlxi5yy7) 30

# ВВЕДЕНИЕ

Предмет изучения раздела «Операционные среды и системное программирование» – принципы построения, организация, основные функции, режимы работы и средства операционных систем, обеспечивающих функционирование современных компьютеров, информационных и вычислительных систем различного назначения.

Целью освоения раздела «Операционные среды и системное программирование» является получение знаний основ построения, функционирования и использования ОС и средств поддержки процессов решения задач на персональном компьютере (ПК) и в среде некоторой прикладной автоматизированной или информационной системы.

Операционные системы могут различаться особенностями реализации внутренних алгоритмов управления основными ресурсами компьютера (процессорами, памятью, устройствами), особенностями использованных методов проектирования, типами аппаратных платформ, областями использования и другими свойствами.

Чтобы разобраться в принципах работы операционной системы, необходимо изучить различные алгоритмы и методы работы с определенными данными. Важно знать, что из себя представляют алгоритмы управления процессами (планирование процессов, организация взаимодействия между процессами, способы борьбы с тупиками), методы распределения памяти и организации файловой системы (иерархия каталогов, способы логической организации, физическая организация).

# 1 ЦЕЛЬ РАБОТЫ

Цель данной лабораторной работы заключается в повторном освоении, укреплении и расширении навыков программирования приложений под операционную систему Windows с фокусом на концепции вычислительных процессов, потоков и нитей, а также их реализации в среде Windows. Предоставляется возможность практически ознакомиться с основными этапами жизненного цикла процессов (потоков) и простым управлением ими, включая порождение, завершение, получение и изменение состояния.

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 рамках выполнения задания предстоит провести сравнительный анализ эффективности (производительности) реализации распараллеливаемого алгоритма умножения матриц с использованием потоков и нитей. Параметры задания предполагают создание потоков и нитей, каждый из которых выбирает свои строки и столбцы матрицы по определенным правилам. После обработки данных каждый поток (нить) выполняет переключение на другой поток с использованием функции Sleep(1).

Основная цель анализа заключается в сравнении результатов выполнения распараллеливаемого алгоритма с "линейной" реализацией без распараллеливания. Также будет проведено соотнесение полученных результатов с уровнем загруженности системы, основываясь на использовании центрального процессора.

Это позволит не только углубить знания по управлению процессами, потоками и нитями в Windows, но и на практике изучить особенности многозадачности и многопоточности при работе с параллельными вычислениями.

# 2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

## 2.1 Управление процессами

Важнейшей частью операционной системы является подсистема управления процессами. Процесс – это программа пользователя в ходе ее выполнения в компьютерной системе. Для операционной системы процесс представляет собой единицу работы, заявку на потребление системных ресурсов.

Подсистема управления процессами планирует выполнение процессов, то есть распределяет процессорное время между несколькими одновременно существующими в системе процессами, а также занимается созданием и уничтожением процессов, обеспечивает процессы необходимыми системными ресурсами, поддерживает взаимодействие между процессами.

## 2.2 Виды потоков

Поток – последовательность команд, обрабатываемых CPU. В рамках одного процесса может находиться один или несколько потоков. Процесс предоставляет ресурсы, поток – команды и данные для обработки. Процесс содержащий один поток называется однопоточным, в противном случае – многопоточным.

Многопоточная модель охватывает 2 категории потоков и их комбинацию:

* потоки на уровне пользователя ULT (User Level Thread);
* потоки на уровне ядра KLT (Kernel Level Thread);
* комбинированная модель UKLT.

ULT управляются самим приложением. KLT управляется самим ядром через интерфейс прикладного программирования средств ядра ОС.

## 2.3 Нити (Fiber)

Нити (Fiber) представляют собой легковесные потоки, которые требуют от приложения управления их переключением. Они работают в контексте потоков, но имеют собственные стек и состояние.

Нити выполняются внутри потоков и могут быть переключены без участия операционной системы, что делает их более эффективными для задач, требующих частого переключения контекста.

Потоки управляются операционной системой, а нити — программным обеспечением на уровне приложения.

Создание нитей выполняется с помощью API, который выделяет необходимый стек и устанавливает контекст исполнения.

Завершение нитей происходит, когда выполнение нити завершается, и ресурсы освобождаются.

Переключение нитей осуществляется вручную в коде приложения, что позволяет изменять активную нить без вмешательства ОС.

## 2.4 Многозадачность

Многозадачность - это когда центральный процессор предоставляется для выполнения нескольких задач одновременно. Многозадачность часто предполагает переключение центрального процессора между задачами, чтобы пользователи могли совместно работать с каждой программой. В отличие от многопоточности, в многозадачности процессы совместно используют отдельные память и ресурсы. Многозадачность предполагает быстрое переключение процессора между задачами, поэтому для переключения от одного пользователя к другому требуется мало времени.

## 2.5 Многопоточность

Многопоточность - это система, в которой множество потоков создаются на основе процесса, посредством которого увеличивается мощность компьютера. При многопоточности процессор предоставляется для одновременного выполнения множества потоков из процесса, а при многопоточности создание процесса выполняется в соответствии с затратами. В отличие от многозадачности, многопоточность предоставляет одинаковую память и ресурсы процессам для выполнения.

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Программа реализует умножение двух матриц размера M x M (где M = 1000) различными методами:

Линейное умножение:

- Функция linenoeYmnozenie() реализует классическое умножение матриц в циклах.

- Алгоритм перебирает элементы каждой ячейки результирующей матрицы и суммирует произведения соответствующих элементов исходных матриц.

Умножение с использованием потоков:

- Функция ymnozeniePotokami() разбивает задачу на N блоков (где N = 100), каждый из которых обрабатывается отдельным потоком.

- Используется mutex для синхронизации доступа к элементам результирующей матрицы, предотвращая гонку данных.

- atomic переменная tasksCompleted отслеживает количество завершенных задач.

- Функция ymnozenieMBlock() выполняет умножение блоков матриц, используя sleep\_for для переключения на другой поток.

Умножение с использованием нитей:

- Функция ymnozenieNitiami() реализует наивное распределение задачи между N нитями.

- Функция использует threadId для передачи управления следующей нити после выполнения блока.

- Используется this\_thread::yield() для предоставления возможности другим нитям получить управление, имитируя работу потоков.

Дополнительные функции:

printMatrix(const vector<vector<int>>& matrix, const string& name): (неиспользуемая в коде) Функция, предназначенная для вывода матрицы на экран с заданным именем.

main():

- Заполняет две матрицы случайными числами.

- Измеряет время выполнения каждого метода умножения.

- Выводит время выполнения каждого метода.

# ЗАКЛЮЧЕНИЕ

В ходе исследования, где сравнивались многопоточные и многонитевые реализации умножения матриц в среде программирования C++ с использованием Visual Studio, были выявлены ключевые аспекты каждого подхода. Эксперименты позволили глубже понять эффективность и особенности работы каждого метода параллельного выполнения задач на примере конкретной операции над данными.

Многопоточные приложения, способные распараллеливать умножение матриц, обладают потенциалом для эффективного использования вычислительных мощностей многопроцессорных систем. Однако, необходимость в сложной синхронизации доступа к общим данным и контроль за потоками операционной системы может повлечь дополнительные затраты времени и ресурсов.

С другой стороны, многонитевые приложения, использующие "легковесные" нити для умножения матриц, могут более эффективно распределить нагрузку между процессорными ядрами без значительных накладных расходов на управление потоками. Это позволяет достичь лучшей отзывчивости и скорости выполнения операций над данными.

Учитывая специфику умножения матриц как критически важной операции во многих областях, результаты исследования могут служить ценным руководством при выборе оптимального подхода к реализации параллельных вычислений в данном контексте. Полученные выводы представляют интерес для разработчиков, стремящихся к оптимизации процессов вычислений и улучшению производительности приложений, особенно при работе с большими объемами данных.

Таким образом, на основе проведенного сравнения многопоточной и многонитевой реализаций умножения матриц в среде C++ с использованием Visual Studio, можно сделать вывод о необходимости индивидуального подхода к выбору метода параллельного выполнения задач в зависимости от конкретных требований проекта и характеристик выполняемых операций. Дальнейшие исследования в этом направлении могут продолжить углубленное изучение эффективных стратегий параллельных вычислений, способствуя развитию инновационных подходов к оптимизации работы с данными и повышению производительности программного обеспечения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

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

[2] Основные сообщения ОС Windows (Win32 API). Программирование в ОС Windows. Лекция 1. – Электронный ресурс. – Режим доступа: https://www.youtube.com/watch?v=wTArIolxch0

[3] Разработка приложений с помощью WinAPI. – Электронный ресурс.– Режим доступа: https://shorturl.at/BDJW8

ПРИЛОЖЕНИЕ А  
(обязательное)  
Исходный код

#include <iostream>

#include <windows.h>

#include <thread>

#include <chrono>

#include <vector>

#include <mutex>

#include <atomic>

#include <ctime>

#include <iomanip>

using namespace std;

const int M = 1000;

const int N = 100;

vector<vector<int>> matrix1(M, vector<int>(M));

vector<vector<int>> matrix2(M, vector<int>(M));

vector<vector<int>> result(M, vector<int>(M));

// Mutex для синхронизации доступа к результатам

mutex resultMutex;

// Атомарные переменные для подсчета завершенных задач

atomic<int> tasksCompleted;

void ymnozenieMBlock(int startRow, int endRow, int startCol, int endCol) {

for (int i = startRow; i < endRow; ++i) {

for (int j = startCol; j < endCol; ++j) {

for (int k = 0; k < M; ++k) {

result[i][j] += matrix1[i][k] \* matrix2[k][j];

}

}

}

this\_thread::sleep\_for(chrono::milliseconds(1));

++tasksCompleted;//счетчик завершенных задач

}

void ymnozeniePotokami() {

int blockSize = M / N;//для обработки потоками

tasksCompleted = 0;

vector<thread> threads;

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

int startRow = i \* blockSize;

int endRow = (i == N - 1) ? M : (i + 1) \* blockSize;

int startCol = i \* blockSize;

int endCol = (i == N - 1) ? M : (i + 1) \* blockSize;

threads.push\_back(thread(ymnozenieMBlock, startRow, endRow, startCol, endCol));

}

for (auto& thread : threads) {//завершениe всех потоков

thread.join();

}

}

void ymnozenieNitiami() {

int threadId = 0;//для передачи управления

int blockSize = M / N;

tasksCompleted = 0;

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

int startRow = i \* blockSize;

int endRow = (i == N - 1) ? M : (i + 1) \* blockSize;

int startCol = i \* blockSize;

int endCol = (i == N - 1) ? M : (i + 1) \* blockSize;

ymnozenieMBlock(startRow, endRow, startCol, endCol);

threadId = (threadId + 1) % N;//передача управления следующей нити

if (threadId == 0) {

this\_thread::yield();

}

}

}

void linenoeYmnozenie() {

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

for (int j = 0; j < M; ++j) {

for (int k = 0; k < M; ++k) {

result[i][j] += matrix1[i][k] \* matrix2[k][j];

}

}

}

}

/\*void printMatrix(const vector<vector<int>>& matrix, const string& name) {

cout << name << ":" << endl;

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

for (int j = 0; j < M; ++j) {

cout << setw(3) << matrix[i][j] << " ";

}

cout << endl;

}

}\*/

int main() {

setlocale(LC\_ALL, "Russian");

srand(time(0));//заполнение матрицы

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

for (int j = 0; j < M; ++j) {

matrix1[i][j] = rand() % 10;

matrix2[i][j] = rand() % 10;

}

}

// Измерение времени выполнения

auto startLinear = chrono::high\_resolution\_clock::now();

linenoeYmnozenie();

auto endLinear = chrono::high\_resolution\_clock::now();

auto durationLinear = chrono::duration\_cast<chrono::milliseconds>(endLinear - startLinear).count();

auto startThreads = chrono::high\_resolution\_clock::now();

ymnozeniePotokami();

auto endThreads = chrono::high\_resolution\_clock::now();

auto durationThreads = chrono::duration\_cast<chrono::milliseconds>(endThreads - startThreads).count();

auto startThreadsNaive = chrono::high\_resolution\_clock::now();

ymnozenieNitiami();

auto endThreadsNaive = chrono::high\_resolution\_clock::now();

auto durationThreadsNaive = chrono::duration\_cast<chrono::milliseconds>(endThreadsNaive - startThreadsNaive).count();

cout << "Размер матриц: " << M << "x" << M << endl;

cout << "Количество потоков/нитей: " << N << endl;

cout << "Время выполнения линейного умножения: " << durationLinear << "мс" << endl;

cout << "Время выполнения умножения потоками: " << durationThreads << "мс" << endl;

cout << "Время выполнения умножения нитями: " << durationThreadsNaive << "мс" << endl;

FILETIME idleTime, kernelTime, userTime;

GetSystemTimes(&idleTime, &kernelTime, &userTime);

auto startCpu = chrono::high\_resolution\_clock::now();

ymnozeniePotokami();

auto endCpu = chrono::high\_resolution\_clock::now();

GetSystemTimes(&idleTime, &kernelTime, &userTime);

auto durationCpu = chrono::duration\_cast<chrono::milliseconds>(endCpu - startCpu).count();// загруженность процессора

auto totalTime = static\_cast<ULONGLONG>(kernelTime.dwHighDateTime) << 32 | kernelTime.dwLowDateTime +

static\_cast<ULONGLONG>(userTime.dwHighDateTime) << 32 | userTime.dwLowDateTime;

auto cpuUsage = static\_cast<double>(totalTime \* 100) / durationCpu;

cout << "Загруженность процессора: " << cpuUsage << "%" << endl;

/\*//Вывод матриц

cout << "Матрица 1:" << endl;

printMatrix(matrix1, "matrix1");

cout << "Матрица 2:" << endl;

printMatrix(matrix2, "matrix2");

cout << "Результат умножения линейным способом:" << endl;

printMatrix(result, "result");

// Сброс результата

result = vector<vector<int>>(M, vector<int>(M));

ymnozeniePotokami();

cout << "Результат умножения потоками:" << endl;

printMatrix(result, "result");

// Сброс результата

result = vector<vector<int>>(M, vector<int>(M));

ymnozenieNitiami();

cout << "Результат умножения нитями:" << endl;

printMatrix(result, "result");\*/

}