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 №2

на тему

**РАБОТА С ФАЙЛАМИ.**

Выполнил: студент гр.253505 Авдошко И.С.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

ОГЛАВЛЕНИЕ

[1 Формулировка задачи 3](#_Toc179742261)

[2 Краткие теоритические сведения 4](#_Toc179742262)

[3 Описание функций программы 5](#_Toc179742263)

[Заключение 6](#_Toc179742264)

[Список использованных источников 7](#_Toc179742265)

[Приложение А (обязательное) 8](#_Toc179742266)

# 1 ФОРМУЛИРОВКА ЗАДАЧИ

Цель лабораторной работы – изучение расширенных возможностей и механизмов работы с файлами и организации ввода-вывода. Также помимо этого в процессе выполнения лабораторной работы будут рассмотрены и изучены неблокирующие и асинхронные операции, мультиплексирование ввода-вывода, отображение файлов в память.

В качестве задачи необходимо разработать программу на языке C++ для обработки данных из файла с использованием механизма отображения файла в память (memory-mapped file). В частности, программа направлена на оценку производительности многопоточной обработки данных при выполнении сортировки в сравнении с традиционным подходом, включающим последовательное чтение, обработку и запись.

Одной из ключевых особенностей является отсутствие мер по предотвращению коллизий при доступе к данным, что позволяет проанализировать влияние параллельного доступа на процесс обработки. Работа сосредоточена на изучении преимуществ и ограничений многопоточности в контексте обработки больших объемов данных через память, что поможет определить оптимальные стратегии для реализации алгоритмов шифрования в таких условиях.

Результаты работы продемонстрируют как преимущества, так и ограничения использования memory-mapped files в современных вычислительных системах.

2 КРАТКИЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

Все программы и данные хранятся в долговременной (внешней) памяти компьютера в виде файлов.

Файл – это определенное количество информации (программа или данные), имеющее имя и хранящееся в долговременной (внешней) памяти.

Имя файла состоит из двух частей, разделенных точкой: имя файла и расширение, определяющее его тип. Имя файлу дает пользователь, а тип файла обычно задается программой автоматически при его создании.

На каждом носителе информации (гибком, жестком или лазерном диске) может храниться большое количество файлов. Порядок хранения файлов на диске определяется установленной файловой системой.

Файловая система – это система хранения файлов и организации каталогов.

Отображение файла в память – это способ работы с файлами в некоторых операционных системах, при котором всему файлу или некоторой непрерывной его части ставится в соответствие определённый участок памяти (диапазон адресов оперативной памяти). При этом чтение данных из этих адресов фактически приводит к чтению данных из отображенного файла, а запись данных по этим адресам приводит к записи этих данных в файл. Отображать на память часто можно не только обычные файлы, но и файлы устройств. [1]

Неблокирующие операции ввода-вывода позволяют программе продолжать выполнение, не дожидаясь завершения операций, что улучшает эффективность использования ресурсов. Если операция не может быть выполнена немедленно, управление возвращается в программу, позволяя ей обрабатывать другие задачи. Асинхронные операции также инициируют ввод-вывод, но программа продолжает работу, не дожидаясь завершения, и получает уведомление о готовности данных через коллбек или событие.

Мультиплексирование ввода-вывода – это техника, которая позволяет управлять несколькими потоками ввода-вывода с помощью одного потока управления. С помощью функций система отслеживает состояние нескольких файловых дескрипторов, определяя, какой из них готов к операциям ввода-вывода. Эти подходы обеспечивают более эффективное управление ресурсами и позволяют обрабатывать множество потоков ввода-вывода без создания большого количества потоков.

Хотя неблокирующие и асинхронные операции, а также мультиплексирование ввода-вывода значительно повышают производительность приложений, работающих с вводом-выводом, их реализация может быть более сложной из-за необходимости управления состоянием и синхронизацией. Тем не менее, эти техники являются важными инструментами для оптимизации работы с большими объемами данных и обеспечения высокой отзывчивости приложений. [2]

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

В программе реализована многопоточная сортировка данных файла с использованием двух подходов: отображения файла в память (memory-mapped) и традиционного ввода-вывода (I/O). Для создания файла, заполненного случайными числами типа double, используется функция записи данных в файл с помощью системной функции WriteFile, которая позволяет записать сгенерированные числа в файл.

Сортировка с использованием memory-mapped файлов организована через функции CreateFileMappingA и MapViewOfFile, которые отображают файл в виртуальную память, что позволяет работать с данными напрямую через указатель. Данные разбиваются на блоки, для каждого из которых создаются потоки через CreateThread. Потоки выполняют сортировку своих блоков, а по завершении их работы программа ожидает с помощью WaitForMultipleObjects. После завершения сортировки данные освобождаются через UnmapViewOfFile, а ресурсы закрываются с помощью CloseHandle.

При сортировке с использованием традиционного ввода-вывода программа открывает файл через CreateFileA, читает данные через ReadFile и записывает отсортированные значения обратно с помощью WriteFile. Для каждой копии файлового дескриптора используется DuplicateHandle для обеспечения многопоточности, а SetFilePointerEx управляет перемещением указателя файла для корректного чтения и записи. После завершения сортировки потоки также ожидаются с использованием WaitForMultipleObjects, а дескрипторы файлов закрываются через CloseHandle.

Программа измеряет время выполнения каждой стратегии сортировки и выводит результаты в консоль. Это позволяет оценить производительность разных подходов. Результат работы программы представлен на рисунке 3.1.

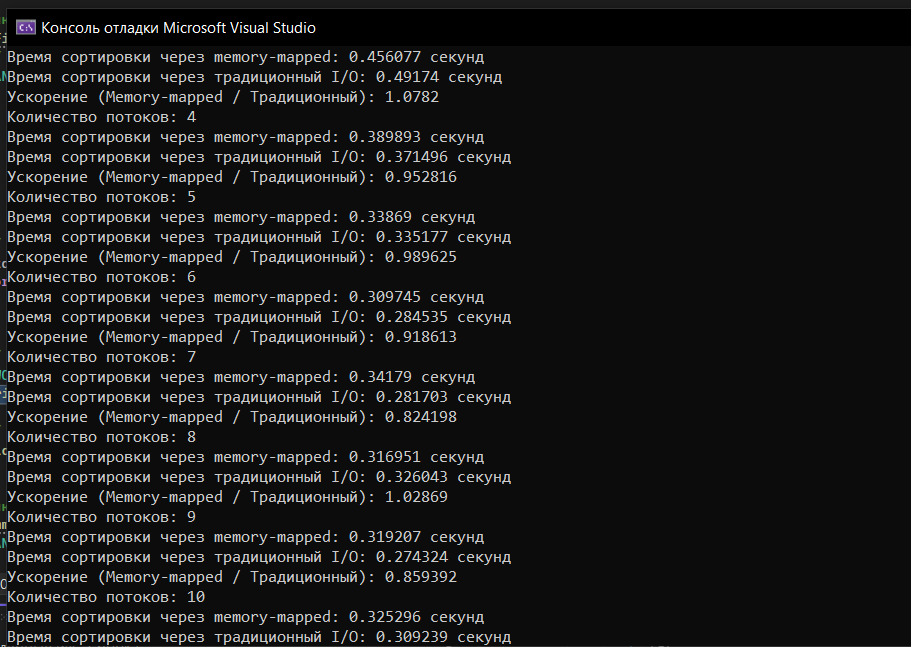


Рисунок 3.1 – Результат работы программы

После нескольких запусков программы можно сделать вывод о том, что коэффициенты скорости выполнения близки к единице, что говорит о незначительной разнице в скорости двух рассматриваемых подходов.

# ЗАКЛЮЧЕНИЕ

В ходе данной лабораторной работы по дисциплине «Операционные системы и системное программирование» была разработана программа для обработки файлов с использованием механизма отображения в память. Основной целью работы стало сравнение эффективности различных подходов к обработке данных, где сортировка использовалась как способ демонстрации производительности при работе от одного до восьми потоков.

В процессе разработки особое внимание уделялось методам управления вводом-выводом и применению системных вызовов и API для работы с файлами и потоками. Это углубило понимание структуры файловых систем и изучение способов оптимизации обработки больших объемов данных.

Лабораторная работа позволила закрепить теоретические знания на практике и глубже разобраться в механизмах работы с процессами и потоками, освоив многопоточную обработку и асинхронные операции. Полученный опыт будет полезен для реализации более сложных проектов в будущем.

В целом, работа подчеркнула важность правильного выбора методов ввода-вывода и их влияние на производительность приложений. Это знание критически важно для системного программирования, так как демонстрирует, как теоретические концепции применяются в реальных условиях, непосредственно влияя на эффективность и функциональность разрабатываемых решений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] StudFiles [Электронный ресурс]. – Режим доступа: https://studfile.net/

[2] Microsoft. "File Management" [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/en-us/windows/win32/fileio/file-management.

ПРИЛОЖЕНИЕ А

(обязательное)

**Исходный код программы**

#include <iostream>

#include <vector>

#include <chrono>

#include <windows.h>

#include <cstdlib>

#include <algorithm>

// Глобальная переменная — количество потоков

size\_t NUM\_THREADS = 12;

// Общее количество чисел для обработки

const size\_t NUM\_DOUBLES = 4 \* 1024 \* 1024;

// Структура для передачи данных в поток при использовании memory-mapped файлов

struct ThreadDataMMap {

double\* start; // Указатель на начало данных для этого потока

size\_t length; // Количество элементов для обработки

};

// Структура для передачи данных в поток при использовании традиционного ввода-вывода

struct ThreadDataTraditional {

HANDLE file; // Файловый дескриптор

LARGE\_INTEGER startOffset; // Начальное смещение в файле (в байтах)

size\_t length; // Количество элементов для обработки

};

// Функция, выполняющая сортировку данных (для memory-mapped файлов)

DWORD WINAPI processChunkMMap(LPVOID param) {

ThreadDataMMap\* data = static\_cast<ThreadDataMMap\*>(param);

// Сортировка данных

std::sort(data->start, data->start + data->length);

return 0;

}

// Функция, выполняющая сортировку данных (для традиционного файлового ввода-вывода)

DWORD WINAPI processChunkTraditional(LPVOID param) {

ThreadDataTraditional\* data = static\_cast<ThreadDataTraditional\*>(param);

// Выделение буфера для данных

std::vector<double> buffer(data->length);

// Перемещение указателя файла на нужное смещение

SetFilePointerEx(data->file, data->startOffset, NULL, FILE\_BEGIN);

// Чтение данных в буфер

DWORD bytesRead;

ReadFile(data->file, buffer.data(), static\_cast<DWORD>(data->length \* sizeof(double)), &bytesRead, NULL);

// Сортировка данных

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

// Перемещение указателя файла обратно к началу

SetFilePointerEx(data->file, data->startOffset, NULL, FILE\_BEGIN);

// Запись отсортированных данных обратно в файл

DWORD bytesWritten;

WriteFile(data->file, buffer.data(), static\_cast<DWORD>(data->length \* sizeof(double)), &bytesWritten, NULL);

// Закрытие дескриптора файла для данного потока

CloseHandle(data->file);

return 0;

}

// Функция для заполнения файла случайными числами типа double

void fillFileWithDoubles(const char\* filename, size\_t numDoubles) {

// Создание файла или перезапись существующего

HANDLE file = CreateFileA(

filename,

GENERIC\_WRITE,

0,

nullptr,

CREATE\_ALWAYS,

FILE\_ATTRIBUTE\_NORMAL,

nullptr);

// Генерация случайных данных

std::vector<double> data(numDoubles);

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

data[i] = static\_cast<double>(rand()) / RAND\_MAX \* 1000; // Числа в диапазоне от 0 до 1000

}

// Запись данных в файл

DWORD bytesWritten;

WriteFile(file, data.data(), static\_cast<DWORD>(numDoubles \* sizeof(double)), &bytesWritten, nullptr);

// Закрытие файла

CloseHandle(file);

}

// Функция для сортировки файла с использованием memory-mapped файлов

void mmapSort(const char\* filename) {

HANDLE file = CreateFileA(

filename,

GENERIC\_READ | GENERIC\_WRITE,

FILE\_SHARE\_READ | FILE\_SHARE\_WRITE,

nullptr,

OPEN\_EXISTING,

FILE\_ATTRIBUTE\_NORMAL,

nullptr);

LARGE\_INTEGER fileSize;

GetFileSizeEx(file, &fileSize);

// Создание объекта отображения файла

HANDLE fileMapping = CreateFileMappingA(

file,

nullptr,

PAGE\_READWRITE,

0,

0,

nullptr);

double\* data = static\_cast<double\*>(MapViewOfFile(

fileMapping,

FILE\_MAP\_ALL\_ACCESS,

0,

0,

0));

// Подсчет количества элементов и разбиение на блоки для потоков

size\_t numDoubles = fileSize.QuadPart / sizeof(double);

size\_t chunkSize = numDoubles / NUM\_THREADS;

std::vector<HANDLE> threads(NUM\_THREADS);

std::vector<ThreadDataMMap> threadData(NUM\_THREADS);

// Создание потоков для сортировки

for (size\_t i = 0; i < NUM\_THREADS; i++) {

size\_t start = i \* chunkSize;

size\_t end = (i == NUM\_THREADS - 1) ? numDoubles : (start + chunkSize);

size\_t length = end - start;

threadData[i] = { data + start, length };

threads[i] = CreateThread(

nullptr,

0,

processChunkMMap,

&threadData[i],

0,

nullptr);

}

// Ожидание завершения всех потоков

WaitForMultipleObjects(static\_cast<DWORD>(NUM\_THREADS), threads.data(), TRUE, INFINITE);

for (HANDLE thread : threads) {

CloseHandle(thread);

}

UnmapViewOfFile(data);

CloseHandle(fileMapping);

CloseHandle(file);

}

// Функция для сортировки файла с использованием традиционного ввода-вывода

void traditionalSort(const char\* filename) {

HANDLE file = CreateFileA(

filename,

GENERIC\_READ | GENERIC\_WRITE,

FILE\_SHARE\_READ | FILE\_SHARE\_WRITE,

nullptr,

OPEN\_EXISTING,

FILE\_ATTRIBUTE\_NORMAL,

nullptr);

LARGE\_INTEGER fileSize;

GetFileSizeEx(file, &fileSize);

size\_t numDoubles = fileSize.QuadPart / sizeof(double);

size\_t chunkSize = numDoubles / NUM\_THREADS;

std::vector<HANDLE> threads(NUM\_THREADS);

std::vector<ThreadDataTraditional> threadData(NUM\_THREADS);

// Создание потоков для сортировки

for (size\_t i = 0; i < NUM\_THREADS; i++) {

size\_t start = i \* chunkSize;

size\_t end = (i == NUM\_THREADS - 1) ? numDoubles : (start + chunkSize);

size\_t length = end - start;

HANDLE threadFileHandle;

DuplicateHandle(

GetCurrentProcess(),

file,

GetCurrentProcess(),

&threadFileHandle,

0,

FALSE,

DUPLICATE\_SAME\_ACCESS);

LARGE\_INTEGER startOffset;

startOffset.QuadPart = start \* sizeof(double);

threadData[i] = { threadFileHandle, startOffset, length };

threads[i] = CreateThread(

nullptr,

0,

processChunkTraditional,

&threadData[i],

0,

nullptr);

}

WaitForMultipleObjects(static\_cast<DWORD>(NUM\_THREADS), threads.data(), TRUE, INFINITE);

for (HANDLE thread : threads) {

CloseHandle(thread);

}

CloseHandle(file);

}

int main() {

setlocale(LC\_ALL, "ru");

const char\* filename = "big\_file.dat";

for (NUM\_THREADS = 1; NUM\_THREADS <= 12; NUM\_THREADS++) {

std::cout << "Количество потоков: " << NUM\_THREADS << '\n';

// Заполнение файла случайными числами

fillFileWithDoubles(filename, NUM\_DOUBLES);

// Измерение времени сортировки с использованием memory-mapped файлов

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

mmapSort(filename);

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

std::chrono::duration<double> durationMmap = endMmap - startMmap;

std::cout << "Время сортировки через memory-mapped: " << durationMmap.count() << " секунд\n";

// Заполнение файла случайными числами заново

fillFileWithDoubles(filename, NUM\_DOUBLES);

// Измерение времени сортировки с использованием традиционного I/O

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

traditionalSort(filename);

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

std::chrono::duration<double> durationTraditional = endTraditional - startTraditional;

std::cout << "Время сортировки через традиционный I/O: " << durationTraditional.count() << " секунд\n";

// Рассчет ускорения

double speedup = durationTraditional.count() / durationMmap.count();

std::cout << "Ускорение (Memory-mapped / Традиционный): " << speedup << '\n';

}

return 0;

}