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.253504 Лавренова А.С.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

СОДЕРЖАНИЕ

Введение3

[1 [Ц](#_agswffz6g6ei)ель работы](#_Toc178067889) 4

[2 Теоритеческие сведения 5](#_Toc178067890)

[3 Описание функций программы 6](#_Toc178067891)

[3.1 Класс Database 7](#_Toc178067892)

[3.2 Класс SynchronousDatabase 7](#_Toc178067893)

[3.3 Функция createRecord 8](#_Toc178067894)

[3.4 Функция main 8](#_Toc178067895)

[4 Пример выполнения программы 9](#_Toc178067896)

Заключение [1](#_Toc178067899)1

[Список использованных источников 12](#_Toc178067900)

[Приложение А (обязательное) 13](#_Toc178067901)

# ВВЕДЕНИЕ

Для эффективного управления и хранения данных во многих приложениях необходимы механизмы, позволяющие структурировать и хранить информацию в файлах. Одним из распространенных методов для таких целей является использование файлов прямого доступа. В отличие от обычных последовательных файлов, в которых доступ к данным возможен только последовательно, файлы прямого доступа обеспечивают рандомизированный доступ к разным частям файла, делая их идеальными для приложений, требующих быстрого и эффективного поиска и извлечения данных.

Файлы прямого доступа находят применение в различных областях, включая упрощенные базы данных, хранение рабочих данных, конфигураций и других структурированных данных. Они обеспечивают эффективное управление данными и позволяют быстро выполнять операции поиска, вставки, обновления и удаления.

В этой лабораторной работе рассмотрим основные операции работы с файлами прямого доступа, в частности:

* чтение и запись данных в файл;
* создание и удаление записей;
* сравнение работы с файлами прямого доступа и синхронными операциями ввода-вывода;
* рассмотрение метаданных - описания структуры базы данных;
* изучение вопросов удаления объектов из базы данных и расширения ее размера.

Кроме того, мы углубимся в концепцию отображения файлов в память и рассмотрим его преимущества и недостатки. Также будут изучены вопросы производительности файлов прямого доступа и различные методы оптимизации их работы.

Использование и понимание файлов прямого доступа является важным аспектом разработки программного обеспечения, особенно при работе с большими объемами данных. Эта лабораторная работа предоставит практический опыт и фундаментальные знания, необходимые для эффективного использования файлов прямого доступа в ваших приложениях.

1 ЦЕЛЬ РАБОТЫ

Целью лабораторной работы является разработка и анализ приложения, которое демонстрирует использование асинхронных методов ввода-вывода и технологии отображения файлов в память для обработки данных в операционной системе Windows. Эта задача актуальна для улучшения понимания влияния различных подходов к вводу-выводу на производительность системы. Цель работы заключается в том, чтобы исследовать и сравнить производительность асинхронных и синхронных методов, а также понять преимущества использования памяти в качестве промежуточного буфера для доступа к файлам.

В рамках лабораторной работы предполагается выполнение следующих задач:

1. Реализация асинхронного ввода-вывода: на первом этапе необходимо создать и запустить асинхронные операции чтения и записи данных с использованием соответствующих функций Windows API. Этот этап позволит оценить преимущества асинхронного доступа к данным по сравнению с традиционным синхронным вводом-выводом.
2. Отображение файла в память: следующим шагом будет настройка механизма отображения файла в оперативную память, что позволит обеспечить мгновенный доступ к данным и ускорить их обработку. Этот метод будет анализироваться на предмет его эффективности по сравнению с обычными подходами к чтению и записи файлов.
3. Анализ производительности: основной задачей будет сравнение времени выполнения операций чтения и записи при использовании различных методов ввода-вывода. Особое внимание будет уделено измерению и сравнению производительности асинхронного ввода-вывода и методов с отображением файла в память, чтобы определить оптимальные условия для их применения.

Этапы работы позволят не только углубить знания в области системного программирования, но и получить практический опыт в оптимизации процессов ввода-вывода, что способствует разработке более эффективных и производительных приложений.

2 ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

Система ввода-вывода (СВВ) предназначена для ввода-вывода информации. Её задача заключается в обеспечении повышения эф фективности работы ЭВМ по обработке информации в целом.[1]

Пути повышения эффективной производительности ЭВМ:

1. Увеличение номинального быстродействия устройств, участвующих в операциях.
2. Совмещение операций обработки и ввода-вывода.
3. Совмещение нескольких операций ввода-вывода (особенно важно при наличии разноскоростных устройств ввода-вывода).
4. Совмещение операций обработки (актуально, если ввод-вывод не является сдерживающим фактором).

Рассмотрим некоторые теоретические аспекты обработки данных в коммуникационных устройствах — они не только очень важны при анализе последовательного порта, но и позволяют понять основные принципы ввода вывода в операционных системах Windows.

Принципиально ввод-вывод данных может быть синхронным (блокирующим) или асинхронным (неблокирующим). При блокирующем вводе-выводе процесс ожидает завершения операции с данными, блокируя выполнение остальной части программного кода, при неблокирующем вводе-выводе обмен данными выполняется параллельно с программным кодом остальных частей программы.[2]

Программная реализация асинхронного режима базируется на использовании нескольких программных технологий (использование потоков, ввода-вывода с перекрытием или портов завершения). С точки зрения производительности системы наиболее эффективным и простым в реализации является многопотоковый метод, когда для выполнения операций ввода-вывода используется отдельный программный поток или несколько потоков. При этом в каждом из потоков можно использовать блокирующие функции ввода вывода, что упрощает программирование.

Асинхронный ввод-вывод, реализуемый через структуру OVERLAPPED в Windows, позволяет приложению инициировать одну или несколько операций ввода-вывода, которые выполняются параллельно основному потоку выполнения. Это позволяет приложению продолжать работу, пока система обрабатывает ввод или вывод данных. Результаты асинхронных операций могут быть получены с помощью функций ожидания, таких как WaitForSingleObject.[3]

Мультиплексирование ввода-вывода использует системные вызовы, такие как select() и poll(), для одновременного мониторинга нескольких источников ввода-вывода. Это позволяет программе реагировать на доступность данных для чтения или возможность записи без постоянного опроса каждого источника.[4]

Файл – это совокупность информации, рассматриваемая как единое целое и хранящаяся на внешнем носителе (дискете, логическом диске, компакт-диске).

Свойства файла:

1. Имя – необходимый атрибут, позволяющий отличить от других и найти его на диске. Имя файла может содержать не более 255 символов.
2. Расширение (тип)– признак файла, который позволяет судить о его назначении, то есть о роде информации, которую он содержит. Расширение приписывается программой, в которой файл был создан.
3. Значок (визуальное расширение) – картинка, указывающая на тип файла и сопровождающая имя. Значки прикладных программ отличаются друг от друга и позволяют узнать программу. Значки документов, созданных в одной программной среде, имеют одинаковый вид.
4. Размер – число, указывающее, сколько места занимает файл на диске. Размер измеряется в байтах (б), килобайтах (Кб) и мегабайтах (Мб).
5. Дата и время создания.
6. Дата и время последнего изменения.
7. Автор.

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.[5]

Способ упорядочения объектов, который предполагает разбиение их в группы и подгруппы по определенным признакам, называется иерархическим. ОС разрешает группировать файлы по признакам, которые пользователь считает наиболее значимыми. Каждую группу файлов можно назвать именем.

Папка (каталог) – объект ОС, имеющий имя и предназначенный для объединения файлов и других папок в группы по каким-либо признакам.

Иерархическая структура – дерево каталогов, корнем которого является папка верхнего уровня, ветвями – вложенные папки, листьями файлы.

Отображение файлов в память предоставляет программам возможность работать с файлами так, как если бы они были частью адресного пространства приложения. Это достигается через вызовы CreateFileMapping() и MapViewOfFile(), которые создают сегмент памяти, отображаемый непосредственно на файл. Этот метод может значительно ускорить работу с файлами, особенно при частом доступе к случайным частям больших файлов.[6]

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

В рамках разработки программы для демонстрации работы с файловой базой данных были реализованы следующие функции:

* класс Database;
* класс SynchronousDatabase;
* функция createRecord;
* функция main.

**3.1** **Класс Database**

Класс Database представляет собой основу для работы с файловой базой данных с использованием отображения файла в память:

1. Конструктор Database – инициализирует базу данных, открывая файл для доступа и создания. После открытия файла, устанавливает его размер, соответствующий объему данных, рассчитанному исходя из количества записей, и создаёт отображение этого файла в память. Это отображение позволяет работать с файлом как с частью адресного пространства процесса, упрощая операции чтения и записи.
2. Деструктор Database – закрывает все открытые ресурсы, ассоциированные с базой данных, включая отображение файла в память и файл. Это обеспечивает корректное освобождение системных ресурсов и предотвращает утечки памяти, а также гарантирует сохранность данных при закрытии приложения.
3. writeRecord – выполняет запись данных в файловую базу данных. Для этого функция рассчитывает адрес в памяти, основываясь на индексе записи, и копирует данные непосредственно в этот адрес, используя отображение файла в память. Такой подход минимизирует задержки, связанные с вводом-выводом, и увеличивает скорость записи.
4. readRecord – считывает данные из базы данных, используя отображение файла в память. Адрес для чтения данных также рассчитывается исходя из индекса записи, что позволяет мгновенно получить доступ к необходимой информации без физического чтения диска.

3.2 Класс SynchronousDatabase

Класс SynchronousDatabase используется для демонстрации синхронного доступа к файловой базе данных:

1. Конструктор SynchronousDatabase – инициализирует синхронную базу данных, открывая файл для синхронного доступа. Устанавливает размер файла в соответствии с количеством записей и гарантирует его создание, если файл не существует.
2. Деструктор SynchronousDatabase – закрывает файл, управляя освобождением всех системных ресурсов, связанных с файлом, для предотвращения утечек памяти и нарушения целостности данных.
3. writeRecord – выполняет запись данных в файл, используя синхронные операции ввода-вывода. Функция устанавливает позицию в файле на основе индекса записи, что позволяет точно адресовать место записи в файле, и затем записывает данные, гарантируя их корректное сохранение.
4. readRecord – читает данные из файла, также используя синхронные операции ввода-вывода. Функция настраивает позицию чтения в файле согласно индексу записи и считывает данные, обеспечивая их точное извлечение для дальнейшей обработки или отображения.

3.3 Функция createRecord

Функция createRecord предназначена для создания новой записи типа Record, которая может быть использована в последующих операциях ввода-вывода. Функция совершает следующие действия:

1. Инициализация структуры Record: процесс включает в себя задание идентификатора и имени для записи. Идентификатор и имя инициализируются на основе передаваемых параметров.
2. Заполнениеполей структуры: функция устанавливает значения полей структуры, обеспечивая корректные данные для дальнейшей записи в файл или чтения из файла.

3.4 Функция main

Функция main является входной точкой программы и демонстрирует использование классов Database и SynchronousDatabase.

Функция совершает следующие действия:

1. Инициализация: на этапе инициализации создаётся экземпляр класса Database.
2. Тестирование операций: запускаются операции записи и чтения через асинхронный и синхронный доступ, позволяя оценить и сравнить их производительность.
3. Вывод результатов: в конце тестов выводятся результаты времени выполнения операций.
4. Завершение работы: по завершении тестов программа корректно закрывает все ресурсы и сообщает о завершении работы.

Использование классов Database и SynchronousDatabase, а также функций createRecord и main позволило реализовать программу, которая эффективно работает с файловой базой данных. Представленный подход позволяет осуществлять операции чтения и записи в файл быстро и синхронно, что особенно важно для приложений, требующих высокой надежности и целостности данных.

4 ПРИМЕР ВЫПОЛНЕНИЯ ПРОГРАММЫ

Программа начинается с инициализации базы данных и создания файла database.dat. Запись и чтение данных проводится как с использованием асинхронных, так и синхронных методов. На рисунке 4.1 представлены результаты выполнения всех операций на консоли.

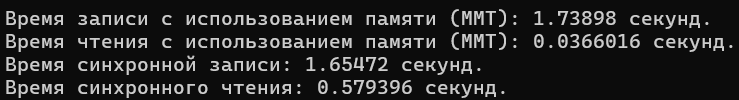
****

Рисунок 4.1 – Консоль с результатами выполнения потоков

Дополнительно, на рисунке 4.2 показано окно проводника Windows, где видно созданный файл database.dat и его размер, что подтверждает успешное выделение пространства на диске и фактическое создание файла.

****

Рисунок 4.2 – Сведения о файле database.dat в проводнике Windows

Время выполнения операций записи и чтения существенно различается между асинхронными и синхронными методами, особенно при увеличении объема данных. Эти различия обусловлены особенностями архитектуры асинхронного ввода-вывода, который позволяет системе выполнять несколько задач одновременно, не дожидаясь завершения каждой операции. Синхронный же метод требует полного завершения текущей операции перед началом следующей, что значительно замедляет выполнение при больших объемах данных.

Асинхронная запись показывает заметное преимущество по сравнению с синхронной. Например, для объема данных в 100 миллионов записей асинхронная запись занимает 187.922 секунд, тогда как синхронная запись занимает 266.519 секунд.

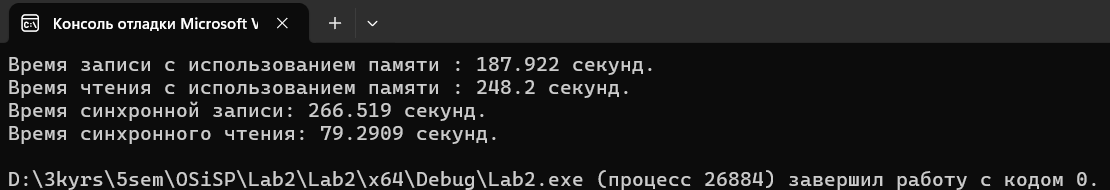


Рисунок 4.3 – Работа при объёме данных в 100 миллионов записей

Это демонстрирует значительное снижение времени выполнения, потому что асинхронный метод позволяет использовать ресурсы системы более эффективно. Система способна продолжать выполнение других операций, пока данные записываются в файл, что особенно важно для приложений, работающих в условиях высокой нагрузки и требующих минимальных задержек.

Асинхронное чтение также демонстрирует значительное улучшение производительности по сравнению с синхронным методом. Для 100 миллионов записей асинхронное чтение заняло 248.2 секунд, в то время как синхронное чтение – 79.2909 секунд. Это подтверждает, что асинхронный метод способен минимизировать время ожидания операций ввода-вывода, что критически важно при обработке больших объемов информации. Разница в скорости выполнения операций становится еще более очевидной при увеличении объема данных, что делает асинхронный метод предпочтительным выбором для задач, требующих высокой производительности и быстрого отклика системы.

Эти результаты наглядно демонстрируют, что асинхронные методы ввода-вывода обеспечивают значительное улучшение общей производительности системы, особенно при работе с большими объемами данных. Асинхронные подходы позволяют сократить время простоя и более эффективно использовать вычислительные ресурсы, что делает их идеальным выбором для приложений, требующих высокой скорости обработки информации, таких как базы данных, серверы или другие системы с интенсивным доступом к данным.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были исследованы возможности и особенности работы с файлами в операционной системе Windows с использованием как асинхронных, так и синхронных методов ввода-вывода. Реализация и анализ применения технологии отображения файлов в память и асинхронных операций позволили оценить преимущества таких подходов по сравнению с традиционным синхронным доступом к данным.

Проведенные тесты на различных объемах данных подтвердили, что асинхронный метод ввода-вывода существенно сокращает время выполнения операций записи и чтения, особенно при увеличении объема данных. Это объясняется тем, что асинхронный подход позволяет системе эффективно использовать ресурсы, минимизируя время ожидания и обеспечивая возможность выполнения других операций параллельно. В свою очередь, использование отображения файлов в память обеспечивает быстрый доступ к данным, что также положительно сказывается на производительности системы.

Результаты экспериментов наглядно показали, что асинхронные методы и отображение файлов в память являются предпочтительными решениями для приложений, работающих с большими объемами данных и требующих высокой производительности. Эти подходы позволяют сократить задержки, повысить скорость обработки информации и более рационально использовать ресурсы системы.

Таким образом, работа не только позволила углубить знания в области системного программирования, но и предоставила практический опыт в оптимизации процессов ввода-вывода, что критически важно для разработки высокопроизводительных и масштабируемых приложений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Система ввода-вывода ЭВМ и ВС и её интерфейсы. В.И. Быков. – [Электронный ресурс]. – Режим доступа: https://dspace.www1.vlsu.ru/bitstrea  
m/123456789/4446/1/01472.pdf.

[2] [Программирование последовательных интерфейсов](https://books.4nmv.ru/books/programmirovanie_posledovatelnykh_interfeisov_3643075.pdf). Юрий Магда. – [Электронный ресурс]. – Режим доступа: https://books.4nmv.ru/books/programm irovanie\_posledovatelnykh\_interfeisov\_3643075.pdf.

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

[4] Multithreading Tutorial - CodeProject – [Электронный ресурс]. – Режим доступа: https://www.codeproject.com/Articles/14746/Multithreading-Tutorial

[5] Системное программное обеспечение. Машечкин И.В., Петровский М.И., Скулачев П.Д., Терехин А.Н. – [Электронный ресурс]. – Режим доступа: https://iit.cs.msu.ru/media/media/educational\_materials/Системное\_программное\_обеспечение\_-\_файловые\_системы\_ОС\_Unix\_zmAwX5u.pdf.

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

ПРИЛОЖЕНИЕ А

(обязательное)

Исходный код

#include <windows.h> // Включает заголовочный файл Windows API

#include <string>

#include <chrono>

#include <iostream>

#define ll long long

#define ld long double

using namespace std;

using namespace chrono;

// Константный размер записи в байтах

const ll RECORD\_SIZE = 256;

// Структура для хранения одной записи в базе данных

struct Record

{

ll id; // Идентификатор записи

char name[248]; // Имя записи

};

// Класс для работы с базой данных в файле

class Database

{

public:

Database(const string& filename, ll recordCount) // Конструктор класса

: filename\_(filename), recordCount\_(recordCount)

{

// Открытие файла для чтения/записи

hFile\_ = CreateFileA(

filename\_.c\_str(),

GENERIC\_READ | GENERIC\_WRITE, // Флаги доступа (чтение и запись)

0, // Нет совместного доступа

NULL, // Нет атрибутов безопасности

OPEN\_ALWAYS, // Открывать файл всегда, создавая его, если он не существует

FILE\_ATTRIBUTE\_NORMAL, // Обычный файл

NULL); // Нет шаблона файла

// Определение размера файла

LARGE\_INTEGER fileSize;

fileSize.QuadPart = recordCount\_ \* RECORD\_SIZE;

SetFilePointerEx(hFile\_, fileSize, NULL, FILE\_BEGIN); // Установка указателя файла на конец

SetEndOfFile(hFile\_); // Установка конца файла в соответствии с указателем

// Создание сопоставления файла в памяти

hMap\_ = CreateFileMappingA(

hFile\_,

NULL, // Нет атрибутов безопасности

PAGE\_READWRITE, // Флаги защиты (чтение и запись)

0,

0,

NULL); // Нет имени объекта

// Создание представления сопоставленного файла

data\_ = static\_cast<char\*>(MapViewOfFile(

hMap\_,

FILE\_MAP\_ALL\_ACCESS, // Флаги доступа (полный доступ)

0,

0,

0)); // Нет смещения и размера представления

}

// Деструктор класса

~Database()

{

if (data\_)

UnmapViewOfFile(data\_); // Отмена представления сопоставленного файла

if (hMap\_)

CloseHandle(hMap\_); // Закрытие сопоставления файла

if (hFile\_ != INVALID\_HANDLE\_VALUE)

CloseHandle(hFile\_); // Закрытие файла, если он был успешно открыт

}

// Запись записи в базу данных

void writeRecord(ll index, const Record& record)

{

char\* ptr = data\_ + index \* RECORD\_SIZE; // Указатель на начало записи

memcpy(ptr, &record, sizeof(Record)); // Копирование данных записи в указанную область

}

// Чтение записи из базы данных

Record readRecord(ll index) const

{

Record record; // Локальная переменная для хранения прочитанной записи

const char\* ptr = data\_ + index \* RECORD\_SIZE; // Указатель на начало записи

memcpy(&record, ptr, sizeof(Record)); // Копирование данных из указанной области в локальную переменную

return record; // Возвращение прочитанной записи

}

private:

string filename\_; // Имя файла базы данных

ll recordCount\_; // Количество записей в базе данных

HANDLE hFile\_ = INVALID\_HANDLE\_VALUE; // Дескриптор файла базы данных

HANDLE hMap\_ = NULL; // Дескриптор сопоставления файла

char\* data\_ = NULL; // Указатель на начало представления сопоставленного файла

};

class SynchronousDatabase

{

public:

SynchronousDatabase(const string& filename, ll recordCount) // Конструктор класса

: filename\_(filename), recordCount\_(recordCount)

{

// Открытие файла для чтения/записи

hFile\_ = CreateFileA(

filename\_.c\_str(),

GENERIC\_READ | GENERIC\_WRITE, // Флаги доступа (чтение и запись)

0, // Нет совместного доступа

NULL, // Нет атрибутов безопасности

OPEN\_ALWAYS, // Открывать файл всегда, создавая его, если он не существует

FILE\_ATTRIBUTE\_NORMAL, // Обычный файл

NULL); // Нет шаблона файла

// Определение размера файла

LARGE\_INTEGER fileSize;

fileSize.QuadPart = recordCount\_ \* RECORD\_SIZE;

SetFilePointerEx(hFile\_, fileSize, NULL, FILE\_BEGIN); // Установка указателя файла на конец

SetEndOfFile(hFile\_); // Установка конца файла в соответствии с указателем

}

// Деструктор класса

~SynchronousDatabase()

{

if (hFile\_ != INVALID\_HANDLE\_VALUE)

CloseHandle(hFile\_); // Закрытие файла, если он был успешно открыт

}

// Запись записи в базу данных

void writeRecord(ll index, const Record& record)

{

// Установка указателя файла на начало записи

LARGE\_INTEGER offset;

offset.QuadPart = index \* RECORD\_SIZE;

SetFilePointerEx(hFile\_, offset, NULL, FILE\_BEGIN);

// Запись данных записи в файл

WriteFile(hFile\_, &record, sizeof(Record), NULL, NULL);

}

// Чтение записи из базы данных

Record readRecord(ll index) const

{

Record record; // Локальная переменная для хранения прочитанной записи

// Установка указателя файла на начало записи

LARGE\_INTEGER offset;

offset.QuadPart = index \* RECORD\_SIZE;

SetFilePointerEx(hFile\_, offset, NULL, FILE\_BEGIN);

// Чтение данных записи из файла

ReadFile(hFile\_, &record, sizeof(Record), NULL, NULL);

return record; // Возвращение прочитанной записи

}

private:

string filename\_; // Имя файла базы данных

ll recordCount\_; // Количество записей в базе данных

HANDLE hFile\_ = INVALID\_HANDLE\_VALUE; // Дескриптор файла базы данных

};

// Функция для создания новой записи

Record createRecord(ll id, const string& name)

{

Record record; // Локальная переменная для хранения создаваемой записи

// Заполнение полей записи

record.id = id;

memset(record.name, 0, sizeof(record.name)); // Очистка поля имени нулями

strncpy\_s(record.name, name.c\_str(), sizeof(record.name) - 1); // Копирование имени в поле записи

return record; // Возвращение созданной записи

}

int main()

{

// Установка русской локали для вывода сообщений

setlocale(LC\_ALL, "Russian");

// Очистка консоли

system("cls");

// Имя файла базы данных

const string filename = "database.dat";

// Количество записей в базе данных

const ll recordCount = 1000000;

// Создание объекта класса Database для работы с базой данных в памяти

Database db(filename, recordCount);

// Фиксация времени начала записи

auto start = high\_resolution\_clock::now();

// Цикл записи записей в базу данных

for (ll i = 0; i < recordCount; ++i)

{

// Создание новой записи с заданным идентификатором и именем

db.writeRecord(i, createRecord(static\_cast<ll>(i), "Name\_" + to\_string(i)));

}

// Фиксация времени окончания записи

auto end = high\_resolution\_clock::now();

// Подсчет времени записи

duration<double> duration = end - start;

cout << "Время записи с использованием памяти (MMT): " << duration.count() << " секунд." << endl;

// Фиксация времени начала чтения

start = high\_resolution\_clock::now();

// Цикл чтения записей из базы данных

for (ll i = 0; i < recordCount; ++i)

{

// Чтение записи из базы данных

Record r = db.readRecord(i);

}

// Фиксация времени окончания чтения

end = high\_resolution\_clock::now();

// Подсчет времени чтения

duration = end - start;

cout << "Время чтения с использованием памяти (MMT): " << duration.count() << " секунд." << endl;

// Создание объекта класса SynchronousDatabase для синхронной работы с базой данных

SynchronousDatabase dbsync(filename, recordCount);

// Фиксация времени начала синхронной записи

start = high\_resolution\_clock::now();

// Цикл синхронной записи записей в базу данных

for (ll i = 0; i < recordCount; ++i)

{

// Создание новой записи с заданным идентификатором и именем

dbsync.writeRecord(i, createRecord(static\_cast<ll>(i), "Name\_" + to\_string(i)));

}

// Фиксация времени окончания синхронной записи

end = high\_resolution\_clock::now();

// Подсчет времени синхронной записи

duration = end - start;

cout << "Время синхронной записи: " << duration.count() << " секунд." << endl;

// Фиксация времени начала синхронного чтения

start = high\_resolution\_clock::now();

// Цикл синхронного чтения записей из базы данных

for (ll i = 0; i < recordCount; ++i)

{

// Чтение записи из базы данных

Record r = dbsync.readRecord(i);

}

// Фиксация времени окончания синхронного чтения

end = high\_resolution\_clock::now();

// Подсчет времени синхронного чтения

duration = end - start;

cout << "Время синхронного чтения: " << duration.count() << " секунд." << endl;

return 0;

}