Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Операционные системы и системное программирование (ОСиСП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту на тему

БИЗНЕС-ЛОГИКА ПРОГРАММНОГО СРЕДСТВА ДЛЯ ДЕФРАГМЕНТАЦИИ

ФАЙЛОВОЙ СИСТЕМЫ NTFS

БГУИР КП I–40 01 01 421 ПЗ

Выполнил

студент: гр. 851004 Пашкевич А.Л.

Проверил: Шульга Е.С.

Минск 2020

Учреждение образования

«Белорусский государственный университет информатики и

радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ПОИТ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Лапицкая Н.В. 2020 г.

ЗАДАНИЕ

по курсовому проектированию

Студенту *Пашкевичу Антону Леонтьевичу* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Тема работы  *Бизнес-логика программного средства для дефрагментации файловой системы NTFS.* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Срок сдачи законченной работы *18.12.2020 г.*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Исходные данные к работе *Интерфейс программирования приложений Windows API, язык программирования C++. Разработать DLL-библиотеку, содержащую функции по дефрагментации файловой системы NTFS. Предоставить API данной библиотеки для использования её сторонними приложениями. Разработать функции по сбору информации о носителе данных и сбору статистики процесса дефрагментации. Предусмотреть возможность работы библиотеки в многопоточной среде. Интегрировать библиотеку в графическое приложение для операционной системы Windows.*

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

*Введение*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*1 Анализ предметной области \_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*2 Анализ требований к программному средству\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*3 Проектирование и разработка программного средства*

*4 Руководство по использованию программного средства \_\_\_\_\_*

*Заключение \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Список использованных источников\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

\_*Приложения* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

*Схема алгоритма дефрагментации файла в формате А1*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. Консультант по курсовой работе  *Шульга Е.С.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

7.Дата выдачи задания *04.09.2020г.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и процентом от общего объема работы):

*Раздел 1. Введение к 15.09.2020г. – 10 % готовности работы;\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 2 к 01.10.2020г. – 30% готовности работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 3 к 30.10.2020г. – 60% готовности работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 4 к 20.11.2020г. – 80% готовности работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Заключение. Приложения к 01.12.2020г. – 90% готовности работы;*

*оформление пояснительной записки и графического материала к 07.12.2020г. – 100% готовности работы.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Защита курсового проекта с 10.12.2020г. по 18.12.2020г.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

РУКОВОДИТЕЛЬ *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Шульга Е.С*

*(подпись)*

Задание принял к исполнению *\_\_\_\_\_\_\_\_\_\_\_\_\_Пашкевич А. Л. 04.09.2020г.*

*(дата и подпись студента)*

Содержание

[Содержание 4](#_Toc58782105)

[Введение 5](#_Toc58782106)

[1 Анализ предметной области 6](#_Toc58782107)

[1.1 Достоинства файловой системы NTFS 6](#_Toc58782108)

[1.2 Структура файловой системы NTFS 7](#_Toc58782109)

[1.3 Дефрагментация диска 9](#_Toc58782110)

[1.4 Обзор существующих аналогов 11](#_Toc58782111)

[2 Анализ требований к программному средству 14](#_Toc58782112)

[3 Проектирование и разработка программного средства 15](#_Toc58782113)

[3.1 Структура программного средства 15](#_Toc58782114)

[3.2 Разработка интерфейсной части 15](#_Toc58782115)

[3.3 Реализация интерфейсных функций 17](#_Toc58782116)

[3.4 Алгоритм дефрагментации 18](#_Toc58782117)

[4 Руководство по использованию программного средства 23](#_Toc58782118)

[Заключение 27](#_Toc58782119)

[Список использованной литературы 28](#_Toc58782120)

[Приложение А 29](#_Toc58782121)

Введение

Системное программное обеспечение – это комплекс программ, которые обеспечивают управление компонентами компьютерной системы, такими как процессор, оперативная память, устройства ввода-вывода, сетевое оборудование, выступая как «межслойный интерфейс», с одной стороны которого аппаратура, а с другой — приложения пользователя. В отличие от прикладного программного обеспечения, системное не решает конкретные практические задачи, а лишь обеспечивает работу других программ, предоставляя им сервисные функции, абстрагирующие детали аппаратной и микропрограммной реализации вычислительной системы, управляет аппаратными ресурсами вычислительной системы.

На сегодняшний день системное программное обеспечение не пользуется такой большой популярностью как, например, в начале 2000-х годов. Это обуславливается тем, что, в основном, оно производится крупными корпорациями, которые являются монополистами, и больший спрос составляют различного рода веб-приложения и веб-службы. Тем не менее, важность системного ПО остаётся прежней, особенно в области разработки новых операционных систем на принципиально новых архитектурах.

Данный курсовой проект направлен на создание динамически подключаемой библиотеки (далее DLL-библиотеки), предоставляющей функции по дефрагментации файловой системы NTFS. Актуальность такого продукта подкрепляется бурным ростом размеров файлов в операционных системах, а также необходимостью рационального использования памяти при их физическом расположении на накопителе.

Помимо реализации функций по дефрагментации, в данном курсовом проекте будет разработан функционал по сбору информации о диске и предусмотрена возможность работы библиотеки в многопоточной среде.

# Анализ предметной области

## Достоинства файловой системы NTFS

NTFS (англ. new technology file system - «файловая система новой технологии») – стандартная файловая система для семейства операционных систем Windows NT фирмы Microsoft.

Для размещения файлов в NTFS используется кластеры. Размер кластера может варьироваться от 512 [байт](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B9%D1%82) до 64 КБ в зависимости от размера тома и версии операционной системы. Для пометки кластеров, как занятых используется по одной битовой карте для каждого тома. С целью улучшения производительности, надёжности и эффективности использования дискового пространства для хранения информации о файлах в NTFS используются специализированные структуры данных. Информация о файлах хранится в главной файловой таблице — Master File Table (MFT). Содержимое папок в NTFS представляется с помощью B± дерева, что является достаточно эффективным способом [1].

Отличительной особенностью данной файловой системы является то, что NTFS поддерживает хранение метаданных и разграничение доступа к данным для различных пользователей и групп пользователей, а также позволяет назначать дисковые квоты (ограничения на максимальный объём дискового пространства, занимаемый файлами тех или иных пользователей). Для повышения надёжности файловой системы в NTFS используется система журналирования USN.

NTFS обладает свойством восстанавливаемости – может возвращаться к работоспособному состоянию после возникновения сбоя. Реализуется такая возможность, во-первых, за счет поддержки атомарных транзакций, во-вторых, за счет избыточности хранения информации. Избыточность используется при хранении важнейших данных файловой системы, критически необходимых для её корректной работы;

Файловая система NTFS появилась в 1993 году заменила файловую систему FAT, использовавшуюся в ОС MS-DOS и ОС Windows и обладает рядом достоинств перед ней. Так, максимальный размер диска, который поддерживает NTFS, составляет в теории байт (16 ЭиБ), а на практике 255 тб, в то время, как FAT поддерживает приблизительно 243 байт (8 [ТиБ](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%B1%D0%B8%D0%B1%D0%B0%D0%B9%D1%82)), что значительно меньше.

Размер тома в FAT ограничивается 4 177 920 кластерами, а с учётом самих таблиц FAT и при максимальном размере кластера 32 Кбайт размер тома может быть до 127.53 ГБ. В NTFS том можно создать размером с весь диск: он будет содержать максимальное количество адресуемых кластеров - при максимальном размере кластера в 64 кб.

NTFS, в отличие от FAT, поддерживает жёсткие и символьные ссылки, соединения для каталогов. Для обеспечения безопасности, помимо атрибутов файлов, использует авторизацию с использованием DACL, шифрование с использованием EFS, аудит с использованием SACL [1].

Максимальный размер файла в NTFS составляет 16 терабайт, в то время как FAT32 может работать с файлами размером не более 4 Гб.

Таким образом, можно сделать вывод, что достоинства NTFS над FAT сводятся к трём пунктам:

* Способность автоматически восстанавливаться после некоторых ошибок диска (FAT не обладает такой способностью).
* Улучшенная поддержка больших объёмов данных, расширение размера диска и файлов, увеличение количества файлов.
* Более высокая степень безопасности. Возможно использование разрешений и шифрования для запрета пользовательского доступа к определенным файлам. В файловой системе FAT все файлы видны абсолютно любому пользователю.

## Структура файловой системы NTFS

В начале тома находится загрузочная запись тома (Volume Boot Record), в которой содержится код загрузки Windows, информация о томе (в частности, тип файловой системы), адреса системных файлов ($Mft и $MftMirr). Загрузочная запись занимает обычно 8 КБ (16 первых секторов).

В определенной области тома (адрес начала этой области указывается в загрузочной записи) расположена основная системная структура NTFS – главная таблица файлов (Master File Table, MFT). В записях этой таблицы содержится вся информация о расположении файлов на томе, а небольшие файлы хранятся прямо в записях MFT.

Важной особенностью NTFS является то, что вся информация, как пользовательская, так и системная, хранится в виде файлов. Имена системных файлов начинаются со знака "$". Например, загрузочная запись тома содержится в файле $Boot, а главная таблица файлов – в файле $Mft. Такая организация информации позволяет единообразно работать как с пользовательскими, так и с системными данными на томе [2].

Поскольку MFT является важнейшей системной структурой, к которой при операциях с томом наиболее часто происходят обращения, выгодно хранить файл $Mft в непрерывной области логического диска, чтобы избежать его фрагментации (размещения в разных областях диска), и, следовательно, повысить скорость работы с ним. С этой целью при форматировании тома выделяется непрерывная область, называемая зоной MFT (MFT Zone). По мере увеличения главной таблицы файлов, файл $Mft расширяется, занимая зарезервированное место в зоне. Остальное место на томе NTFS отводится под файлы – системные и пользовательские.

На рисунке 1.1 показаны структуры данных, связанные с NTFS.

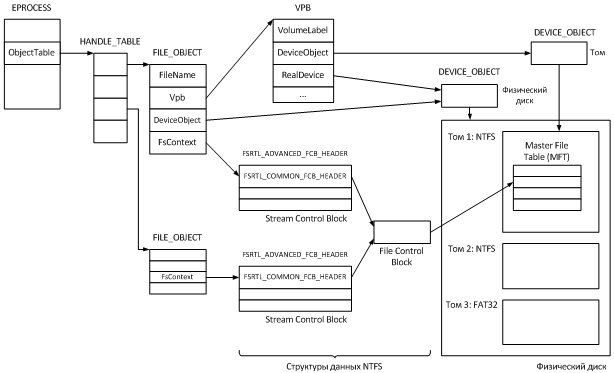


Рисунок 1.1 – Структуры данных, связанные с NTFS

Основная информация о файле содержится в файловой записи размером 1 КБ таблицы MFT, а небольшие файлы целиком хранятся в файловой записи.

Файловая запись состоит из заголовка и набора атрибутов. В заголовке содержится служебная информация о файловой записи, например, её тип и размер. Все данные, относящиеся непосредственно к файлу, хранятся в виде атрибутов. Названия атрибутов, так же как и системных файлов, начинаются с "$". Например, отдельными атрибутами являются имя файла ($FILE\_NAME), информация о его свойствах ($STANDARD\_INFORMATION), данные файла ($DATA).

На диске файловая запись всегда расположена в начале сектора, первые байты файловой записи кодируют слово "FILE". Конец записи определяется 4 байтовой последовательностью FF FF FF FF [2].

Физически атрибут файла хранится в виде потока байтов – простой последовательности байтов. Такое представление позволяет одинаковым образом работать с разнотипными атрибутами, а также добавлять нестандартные пользовательские атрибуты. Каждый атрибут состоит из заголовка, определяющего тип атрибута и его свойства, и тела, содержащего основную информацию атрибута. Структура файловой записи представлена на рисунке 1.2.

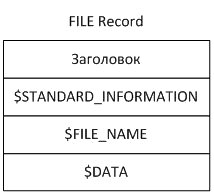


Рисунок 1.2 – Файловая запись

## Дефрагментация диска

Существует два типа фрагментации дискового пространства: внутренняя и внешняя. Внутренняя фрагментация наблюдается тогда, когда фактический размер выделенного блока памяти для файла больше чем нужный, и много памяти внутри блока не используется. Например, файл размером 512 байт занимает целый кластер в 4 Кб, при этом оставшаяся память не может использоваться файловой системой.

Внешняя фрагментация появляется тогда, когда есть достаточно большой объём свободной памяти для выделения блока, но нет непрерывного участка достаточного размера. В этом случае новый файл разбивается на фрагменты различной величины для записи в различные (не последовательные) области жёсткого диска, то есть становится фрагментированным.

Внешняя фрагментация файловой системы, по своей сути, является компромиссом между простотой и скоростью первоначальной записи данных на диск и скоростью чтения этих же самых данные в последующем. Данное явление может происходить с связи с активным использованием файловой системы. Например, при удалении файлов образуется свободный блок памяти, которого может быть недостаточно в последующем для записи большего файла, тогда он будет фрагментирован. Также при увеличении размера файла сверх размера кластера файловая система помещает оставшуюся часть файла в конце дискового пространства – файл снова оказывается фрагментированным.

К основным последствиям внешней фрагментации относятся уменьшение скорости работы накопителя с фрагментированными файлами, поскольку возникают задержки на физическое перемещение головки диска между разделёнными фрагментами, а также замедляется запись новых файлов на накопитель.

По этим причинам используется дефрагментация диска – процесс перераспределения фрагментов файлов и логических структур файловой системы для обеспечения непрерывной последовательности кластеров.

В случае использования жестких дисков, вследствие дефрагментации ускоряется чтение и запись файлов, а также работа программ и операционной системы. Это достигается за счет того, что чтение файлов после дефрагментации производится максимально линейно и непрерывно, без дополнительных передвижений головки жёсткого диска для поиска и воссоединения фрагментов.

Твердотельные накопители на основе различных видов флеш-памяти в дефрагментации файловых систем не нуждаются. Флеш-память является памятью с прямым доступом, в которой время доступа к любой ячейке сравнительно мало и одинаково вне зависимости от её адреса. Более того, дефрагментация, в некоторой степени, вредит флеш-памяти, так как она имеет намного более ограниченное количество циклов записи/перезаписи, чем накопители на магнитных дисках [3].

Существует несколько типов дефрагментации:

* Простая дефрагментация. Дефрагментируются только фрагментированные файлы, свободное же место не обрабатывается. Ускоряется работа с уже существующими файлами.
* Дефрагментация свободного пространства. Обрабатывается только свободное пространство, файлы же остаются фрагментированными. Намного ускоряет запись новых файлов.
* Полная дефрагментация. Одновременно дефрагментируется и свободное пространство, и сами файлы, которые и помещаются в начало раздела, что позволяет освободить максимально возможную свободную область диска. Ускоряет работу как с новыми файлами, так и с уже существующими [3].
* Оптимизация. Помимо дефрагментации перемещает физически в начало раздела мелкие и важные для работы системы файлы и перемещает в конец диска большие, пользовательские или редко используемые.

Алгоритм простой дефрагментации является самым часто используемым и наиболее простым в реализации, быстрым среди остальных алгоритмов. Проводится анализ файла и поиск всех кластеров этого файла. После составления карты кластеров проводится поиск подходящего места для расположения файла целиком и перемещение его фрагментов в этой место. Если же место для файла не найдено, то он пропускается и остается фрагментированным. Результат алгоритма простой дефрагментации схематично представлен на рисунке 1.3.

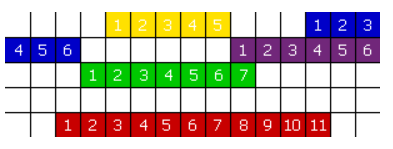


Рисунок 1.3 – Простая дефрагментация

Алгоритм полной дефрагментации в настоящее время используется крайне редко, как основной метод, так как он занимает длительное время и требует относительно много свободного дискового пространства. В отличие от простой дефрагментации, перемещение файлов осуществляется не в любое подходящее место на диске, а в конец раздела, таким образом, образуется область данных без свободных кластеров. После дефрагментации всех файлов выполняется их перемещение в свободные кластеры в начале тома. Таким образом проводится реорганизация файловой структуры, благодаря которой в начале дискового раздела создается область данных, а в конце – все доступное свободное дисковое пространство [4]. Схематично результат алгоритма полной дефрагментации представлен на рисунке 1.4.

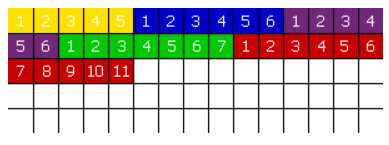


Рисунок 1.4 – Полная дефрагментация

## Обзор существующих аналогов

Почти во всех современных операционных системах существуют встроенные утилиты, выполняющие функцию дефрагментации дискового пространства. Однако, зачастую, они реализуют не самые оптимальные алгоритмы и предоставляют скудный функционал. В этой связи большой интерес представляют средние и крупные коммерческие проекты. Одним из таких приложений является IObit Smart Defrag.

Данное программное средство не только дефрагментирует сами файлы, но и распределяет их на диске по признаку частоты использования. Все операции можно настроить для их выполнения в автоматическом режиме. Разработчики утверждают, что Smart Defrag использует одни из самых эффективных алгоритмов дефрагментации [5]. Есть возможность дефрагментировать отдельные папки или файлы, а также возможность работы по расписанию. Существует опция дефрагментации диска перед загрузкой операционной системы. Приложение имеет продуманный и удобный интерфейс и обеспечивает высокую безопасность при работе, в том числе при перебоях электропитания. Вид окна Smart Defrag, отражающего состояние диска, представлен на рисунке 1.5.

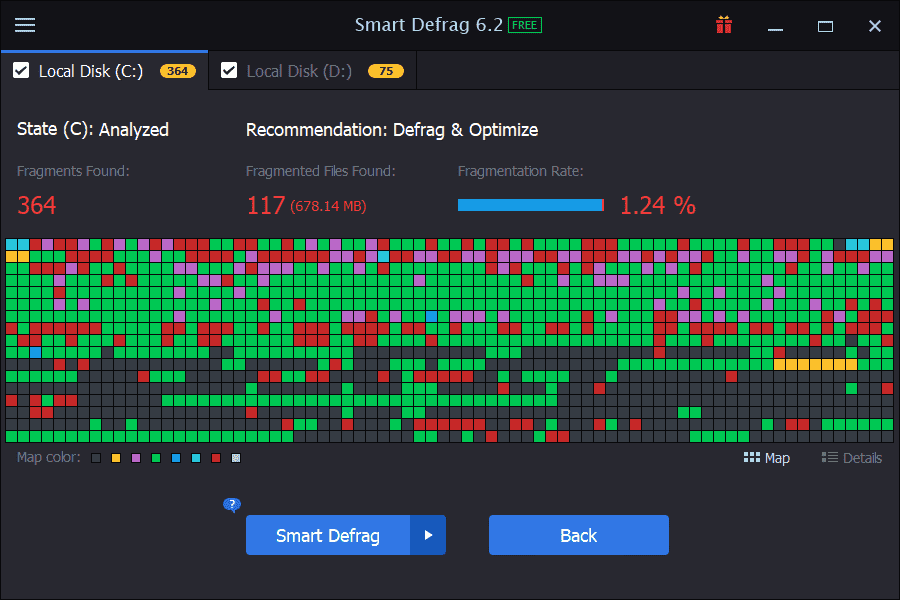


Рисунок 1.5 – Отображение состояния диска в Smart Defrag

Более простым приложением для быстрой оценки и дефрагментации жестких дисков является Defraggler. Данное приложение создано британской частной фирмой Piriform Limited и написано на языке программирования C++. Это бесплатная утилита имеет все самые необходимые функции: оценка состояния диска, анализ, дефрагментации отдельных файлов. Реализована как полный, так и простой алгоритм дефрагментации. Есть возможность автоматически определять тип носителя (HDD, SSD), что очень удобно если подключено несколько носителей. К тому же, как было сказано ранее, дефрагментировать SSD не рекомендуется. Существует портативная версия, не нуждающаяся в установке (программу можно скопировать на флеш-накопитель/внешний HDD и использовать на любом компьютере) [5]. После анализа диска можно просмотреть список фрагментированных файлов. Утилита поддерживает файловые системы NTFS и FAT32, а также все самые современные версии операционной системы Windows. Вид окна Defraggler, отражающего состояние диска, представлен на рисунке 1.6.

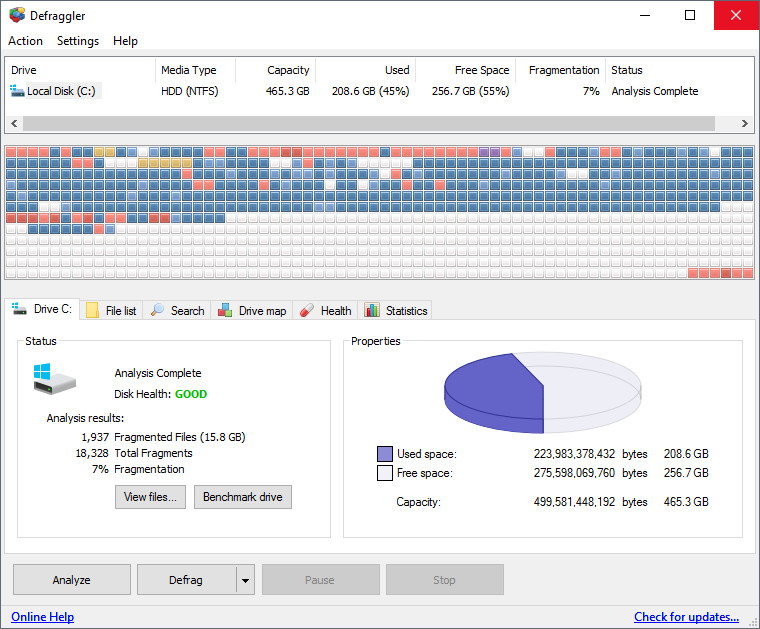


Рисунок 1.6 – Отображение состояния диска в Defraggler

В результате анализа существующих аналогов можно утверждать, что большинство крупных программных средств для дефрагментации предоставляют приблизительно одинаковый функционал с точки зрения пользователя. В него входит полное отображение пользователю информации о дисковой подсистеме, а также о ходе процесса дефрагментации, гибкие настройки управления временными и пространственными характеристиками дефрагментатора, а также возможность выполнения в автоматическом режиме. Однако, стоит заметить, что каждое программное средство реализует алгоритмы дефрагментации по-своему, лишь заявляя о их высокой эффективности. Вместе с тем, детали реализации остаются скрытыми коммерческой тайной, поэтому провести сравнительный анализ бизнес-логики дефрагментаторов не представляется возможным.

# Анализ требований к программному средству

DLL-библиотека DefragmenterCore предназначается для предоставления функций по дефрагментации файловой системы NTFS и сбору информации о дисковой подсистеме.

Интерфейсная часть (API) библиотеки должна содержать следующие элементы:

* структуру данных для хранения информации о диске, включая общий объём памяти на нём и объём свободной памяти;
* функцию получения информации о всех дисках на устройстве;
* структуру данных для хранения информации файловой системы о диске, включая количество секторов и кластеров на диске, их размерность в байтах и расположение таблицы MFT;
* функцию получения информации файловой системы о диске;
* функцию валидации пользователя библиотеки как администратора системы;
* структуры данных для хранения информации о результатах процесса дефрагментации и статистики по обработанным файлам;
* функции получения информации о результатах процесса дефрагментации и статистики по файлам;
* функцию входа в рабочий цикл дефрагментатора;
* функцию остановки процесса дефрагментации диска;
* функции по обеспечению работы библиотеки в многопоточной среде.

Часть реализации библиотеки должна соответствовать следующим требованиям:

* реализовывать алгоритм простой дефрагментации, не затрагивая свободное дисковое пространство;
* журналировать обработку файлов и собирать статистику обработки;
* обеспечивать потокобезопасность библиотеки посредством использования примитивов синхронизации.

Реализация DLL-библиотеки может использовать типы данных языка C++ стандарта ISO/IEC 14882:2011 и типы данных Windows API, а для взаимодействия с файловой системой – только функции Windows API.

# Проектирование и разработка программного средства

## Структура программного средства

Как и любая другая DLL-библиотека DefragmenterCore состоит из интерфейсной части, представленной заголовочным файлом Core.h, и части реализации. Она имеет следующую структуру файлов:

* CheckUser.cpp – реализация функции валидации пользователя библиотеки, как администратора системы;
* Core.cpp – точка входа в DLL-библиотеку;
* Defragger.cpp – реализация дефрагментации, процесса её журналирования и сбора статистики;
* DriveOperation.cpp – реализация получения информации файловой системы о диске;
* Menu.cpp – реализация получения базовой информации о диске.

Как видно, структура программного средства довольно простая, что устраняет трудности в его использовании и переработке. После сборки библиотека компилируется в выходной dll-файл, который непосредственно подключается пользователем с помощью динамического либо статического связывания.

## Разработка интерфейсной части

При разработке DLL-библиотеки с помощью директив условной компиляции необходимо объявить конструкцию, представленную на рисунке 3.1.

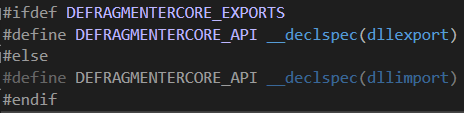


Рисунок 3.1 – Объявление константы DEFRAGMENTERCORE\_API

Впоследствии каждая интерфейсная функция будет объявляться с константой DEFRAGMENTERCORE\_API, что позволит библиотеке экспортировать эту функцию, а стороннему приложению, соответственно импортировать её. Помимо этого, функции будут иметь соглашение вызова cdecl, что объясняется трудностями в применении соглашения вызова stdcall.

Структура DriveData содержит информацию о диске. В ней хранятся буква диска, его тип (фиксированный, отсоединяемый, сетевой, CD-ROM и ОЗУ-диск), количество секторов в кластере и количество байтов в секторе, полный объём памяти на диске и объём свободного пространства. Все поля являются строками типа wchar\_t фиксированного размера. Это сделано для прямой выдачи данных пользователю в виде, удобном для непосредственного использования.

Соответственно, функция getDrives, предназначается для получения базовой информации о дисках. Она возвращает вектор структур DriveData.

Структура DriveInfo содержит информацию файловой системы NTFS о диске. В ней находятся поля для хранения буквы диска, серийного номера тома, количество секторов и кластеров, количество свободных и зарезервированных кластеров, количество байт в секторе и кластере, а также информацию о таблицы MFT: длину мастер-таблицы, номер начального кластера таблицы, номера начального и конечного кластера зоны мастер-таблицы. Все поля представляются строками типа wchar\_t фиксированного размера.

Соответственно, функция getDriveInfo по переданной ей букве диска возвращает структуру DriveInfo с информацией файловой системы о нём.

Точкой входа в рабочий цикл дефрагментации является функция WorkIn. Она использует соглашение вызова stdcall, что необходимо для её передачи в отдельно работающий поток функцией CreateThread. В качестве аргумента она принимает указатель LPVOID на структуру StartDefragInfo, содержащую информацию о начальном месте дефрагментации. В ней содержится текущая директория, буква диска и флаг, обозначающий, является ли директория корневой.

Функции InitCS и DeleteCS вызываются пользователем и обеспечивают инициализацию и удаление внутренней критической секции дефрагментатора.

Функция StopDefrager принимает handle потока, в котором работает дефрагментатор, и останавливает процесс дефрагментации.

Функция getDefragmentationStats может быть вызвана в любой момент времени и возвращает структуру DefragmentationStats. В ней хранится количество пропущенных, дефрагментированных и файлов, при дефрагментации которых произошла ошибка (например, не было найдено свободного участка памяти достаточного размера).

Структура DefragmentationLogItem является элементом журналирования и содержит полный путь к файлу и результат его обработки. Эти поля также представляются строками типа wchar\_t фиксированного размера.

Функция getDefragmentationLogs возвращает очередь таких структур. Она должна вызываться пользователем библиотеки регулярно, чтобы была возможность своевременно отображать результаты процесса дефрагментации. Для регулярного вызова возможно использование события таймера WM\_TIMER и функции SetTimer, в которую можно передать требуемый период посылки сообщения.

Таким образом, API библиотеки предоставляет пользователю необходимый контроль над процессом дефрагментации диска, а также инструменты получения оперативной информации.

## Реализация интерфейсных функций

Для валидации пользователя в качестве администратора системы разработана функция isUserAdmin. Она использует вызов AllocateAndInitializeSid, который получает значение sid группы администратора, а затем с помощью CheckTokenMembership проверяет наличие в ней значения sid текущего пользователя.

В основе реализации функции getDrives лежит вызов функции GetLogicalDriveStringsA. Она позволяет получить буфер, который содержит буквы всех логических дисков, находящихся в системе. Важную роль играет функция GetDriveTypeA, которая по букве диска может вернуть его тип. Функция GetDiskFreeSpaceA по букве диска возвращает количество байтов в секторе и секторов в кластере, а также общее число и число свободных кластеров. Далее, с помощью вычислений находятся показатели из структуры DriveData.

Для представления данных в строках используется функция std::to\_wstring, а также запись в поток std::wostringstream с последующим изъятием из него строки в кодировке Юникод.

Для реализации функции getDriveInfo необходимо открыть файл диска с помощью CreateFileA. При этом необходимо указать атрибуты GENERIC\_READ, GENERIC\_WRITE, FILE\_SHARED\_READ, FILE\_SHARED\_WRITE и OPEN\_EXISTING. Далее выделяется память для буфера с информацией файловой системы NTFS\_VOLUME\_DATA\_BUFFER и выполняется вызов DeviceIoControl. В него передаётся handle дискового файла, константа FSCTL\_GET\_NTFS\_VOLUME\_DATA и указатель на данный буфер. В результате из него можно изымать данные и помещать в структуру DriveInfo. При этом, конвертация в строковое представление осуществляется посредством записи в поток std::wostringstream.

Стоит обратить внимание на функцию DeviceIoControl, так как она будет использоваться в дальнейшем, в том числе в реализации алгоритма дефрагментации. Данная функция выполняет отправку управляющего кода на драйвер устройства, принуждая устройство выполнить данную операцию [6]. В getDriveInfo устройство (диск) идентифицируется значением handle открытого файла, а константа FSCTL\_GET\_NTFS\_VOLUME\_DATA обеспечивает извлечение информации о томе файловой системы NTFS.

Входная функция WorkIn обнуляет статистику по обработанным файлам, распаковывает структуру StartDefragInfo и передаёт управление рабочему циклу – функции Defrag. Функция StopDefragger выставляет флаг остановки и ожидает поток с помощью вызова WaitForSingleObject.

Для организации журналирования реализована функция createLog. Она помещает в хвост очереди новый элемент журналирования, в который записывает результат обработки файла и полный путь к нему. При этом преобразование строк происходит при использовании потока std::wostringstream. Соответственно, интерфейсная функция getDefragmentationLogs возвращает копию данной очереди, а исходную очищает. Также внутри функции createLog происходит вызов createStat для обновления статистики по обработанным файлам.

Следует заметить, что код в двух последних функциях выполняется внутри критической секции. Это необходимое условие для синхронизации потока дефрагментатора и потока, который выполняет забор информации об обработанных файлах. В противном случае, есть большая вероятность потери данных или возникновения исключительных ситуаций. Критическая секция является внутренней, и вызывающему потоку напрямую недоступна. Однако, стоит сказать, что в библиотеке предусмотрены интерфейсные функции её инициализации и удаления.

## Алгоритм дефрагментации

Рабочий цикл дефрагментатора – функция Defrag, в общем, представляет собой рекурсивный проход по папкам требуемого диска. Используется структура WIN32\_FIND\_DATA, вызовом FindFirstFile в неё записывается информация о найденном файле, при этом возвращается его handle. Далее в этой структуре в атрибутах файла проверяется наличие константы FILE\_ATTRIBUTE\_DIRECTORY и, если это папка, происходит рекурсивный вызов Defrag для неё. Если же найденный элемент – файл, то происходит получение битовой карты файла на диске функцией readFileBitmap, которая будет описана позже. Результат помещается в структуру RETRIEVAL\_POINTERS\_BUFFER. Далее с помощью её поля ExtentCount можно проверить, является ли файл фрагментированным, и, если является, функцией Move провести его дефрагментацию. В этом месте цикла, располагается журналирование обработки файлов. Таким методом с помощью функции FindNextFile обрабатываются все файлы директории. Также проверяется флаг остановки для прекращения работы дефрагментатора. Схема алгоритма Defrag представлена на рисунке 3.2.

Вышеупомянутая функция readFileBitmap отвечает за получение битовой карты файла, т.е. карты размещения его на кластерах диска. В начале происходит открытие файла вызовом CreateFile. Указываются атрибуты FILE\_SHARE\_READ, FILE\_SHARE\_WRITE и OPEN\_EXISTING. Далее в цикле происходит получение частей файла с записью их битовую карту. Для этого используется вызов DeviceIoControl с передачей в него значения handle файла и управляющего кода FSCTL\_GET\_RETRIEVAL\_POINTERS. За один вызов возвращается один кластер. Таким образом, следующее чтение должно начинаться с места окончания предыдущего. Эти смещения файла хранятся и модифицируются в структуре STARTING\_VCN\_INPUT\_BUFFER. В результате получается полная битовая карта файла. Схема алгоритма readFileBitmap показана на рисунке 3.3.

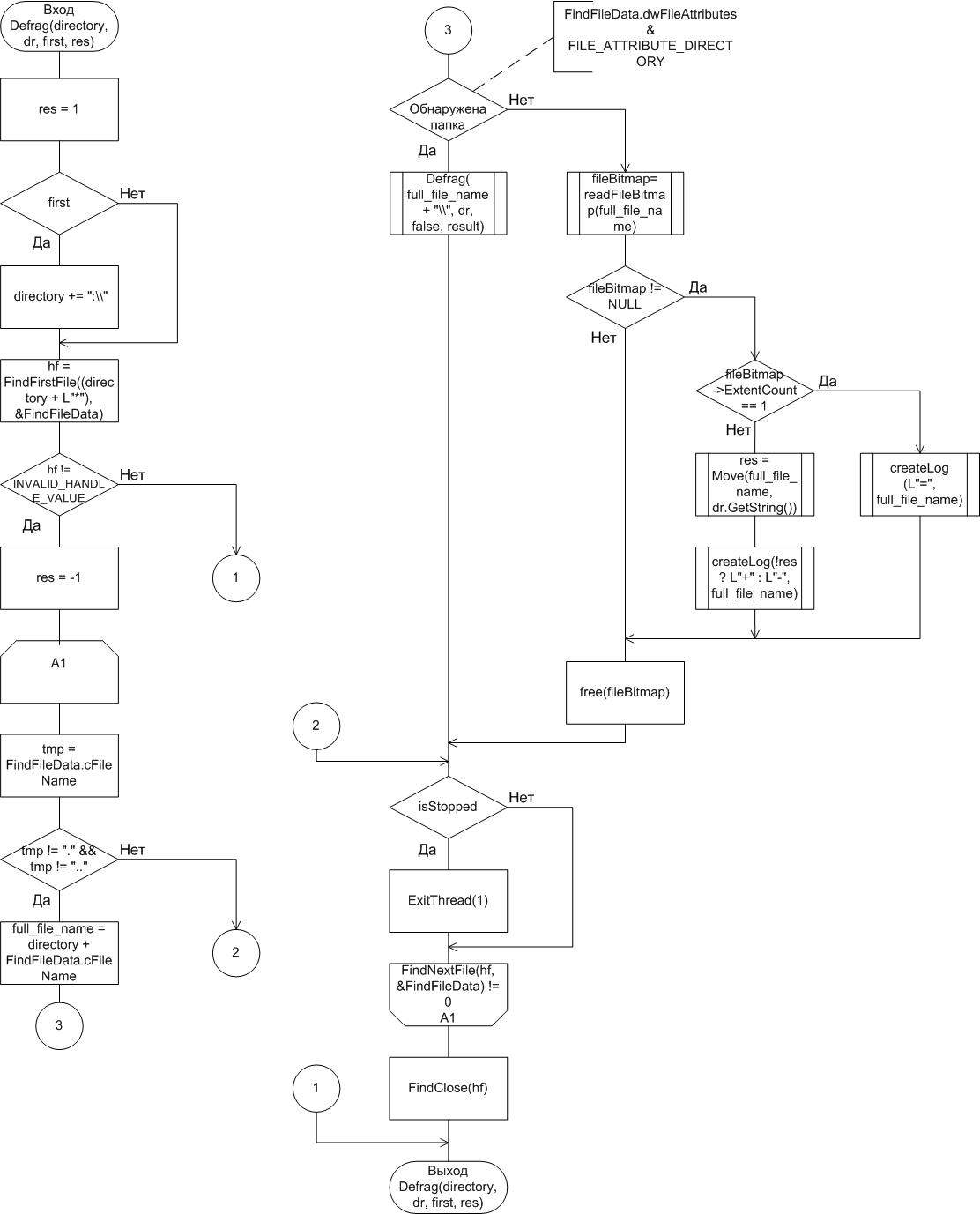


Рисунок 3.2 – Схема алгоритма Defrag

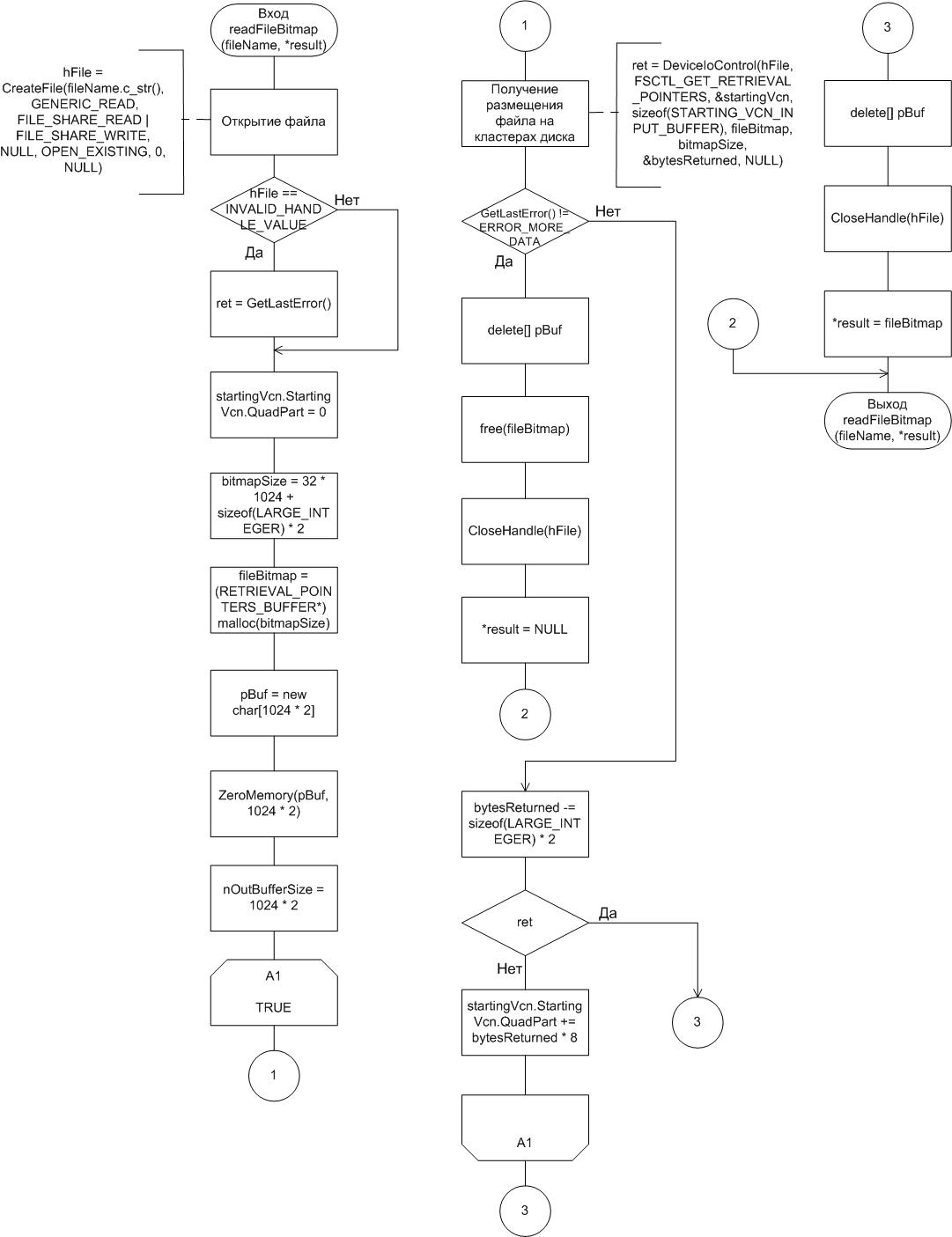


Рисунок 3.3 – Схема алгоритма readFileBitmap

Также в реализации алгоритма дефрагментации важнейшую роль играет информация о свободных и занятых кластерах диска. Для этого разработана функция readVolumeBitmap, которая вызывается каждый раз при необходимости дефрагментации очередного файла. Она получает битовую карту диска и помещает её в структуру VOLUME\_BITMAP\_BUFFER. Алгоритм почти идентичен readFileBitmap: открывается файл диска с помощью CreateFile, вызывается DeviceIoControl с управляющим кодом FSCTL\_GET\_VOLUME\_BITMAP. Однако стоит заметить, что DeviceIoControl вызывается дважды, так как первый вызов возвращает ошибку ERROR\_MORE\_DATA и размер битовой карты диска. Поэтому для записи в буфер результата необходимо выделить объём памяти, равный размеру битовой карты и выполнить второй вызов. Схема алгоритма readVolumeBitmap отображена на рисунке 3.4.

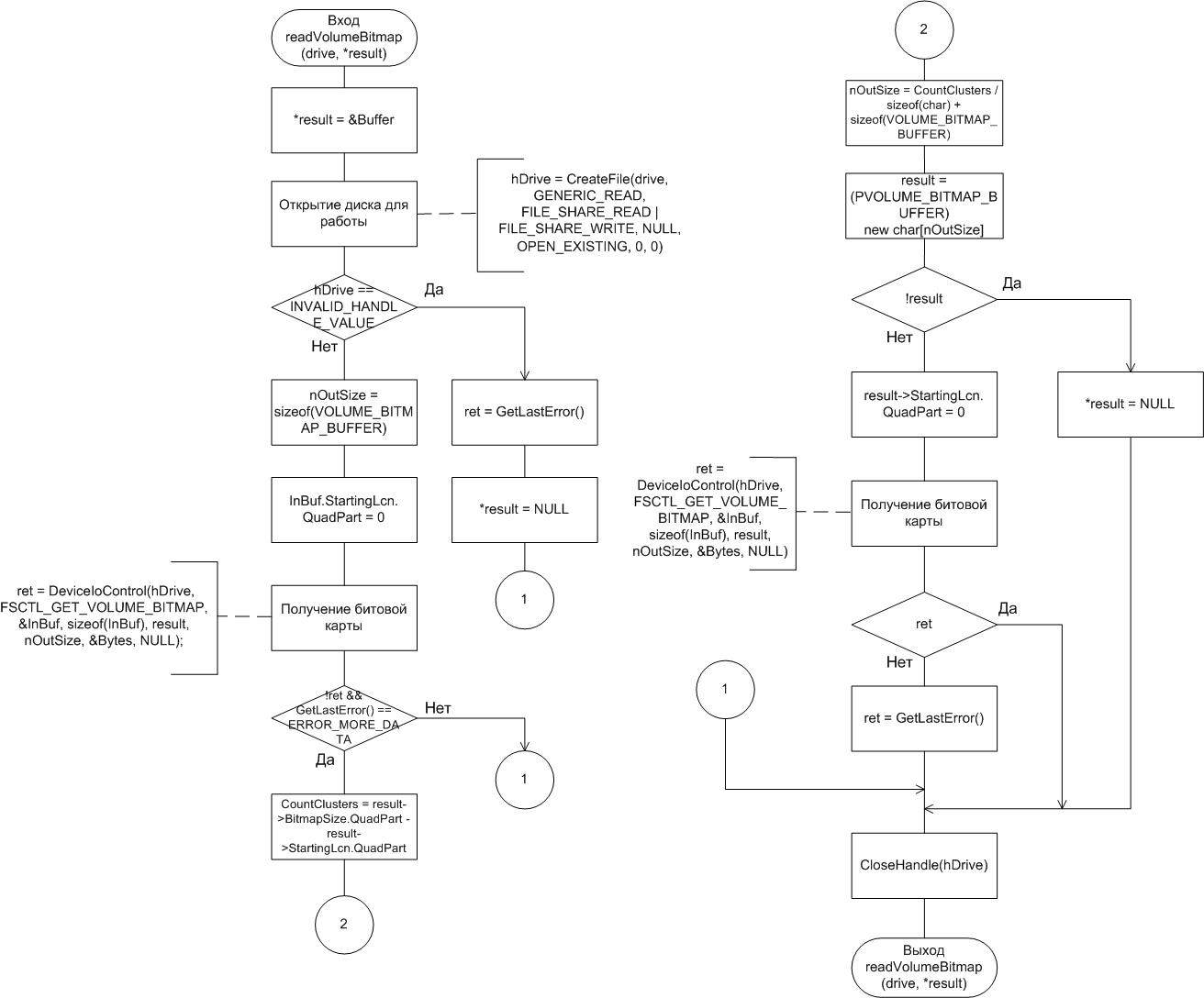


Рисунок 3.4 – Схема алгоритма readVolumeBitmap

Функция Move играет ключевую роль во всей библиотеки и отвечает непосредственно за дефрагментацию файла. Она используется тогда, когда посредством поля ExtentCount структуры RETRIEVAL\_POINTERS\_BUFFER с битовой картой файла было обнаружено, что файл фрагментирован.

На первом этапе с помощью функции readVolumeBitmap необходимо получить битовую карту диска, которая обозначает свободные и занятые кластеры. Далее происходит открытие целевого файла посредством CreateFile. Затем с помощью вызова GetDiskFreeSpace необходимо получить число байтов в секторе и секторов в кластере. Таким образом определяется размер кластера. Далее находится размер целевого файла, и исходя из него выделяется память под результирующий буфер для кластеров файла.

На втором этапе необходимо получить битовую карту файла вызовом функции DeviceIoControl с кодом FSCTL\_GET\_RETRIEVAL\_POINTERS. Далее читаются кластеры файла, и формируется массив с их номерами.

На третьем этапе снова происходит подключение к файлу и диску. Производится анализ битовой карты диска и поиск последовательного места достаточного размера для расположения файла. Номер начального кластера записывается в переменную nStartLCN.

На последнем этапе в цикле происходит перемещение кластеров файла в указанную область. Создаётся структура типа MOVE\_FILE\_DATA, описывающая целевой буфер. В ней указывается начало свободного пространства из переменной nStartLCN и handle целевого файла. Существует управляющий код FSCTL\_MOVE\_FILE, который можно использовать для перемещения кластеров. Это снова осуществляется посылкой сообщения драйверу устройства при помощи DeviceIoControl.

Исходный код функции Move приведён в приложении A. Схема алгоритма дефрагментации файла присутствует в графических документах.

# Руководство по использованию программного средства

Разработанная DLL-библиотека предназначается для предоставления функций по дефрагментации файловой системы NTFS и сбору информации о дисковой подсистеме. Интерфейсные функции библиотеки представлены на рисунке 4.1.

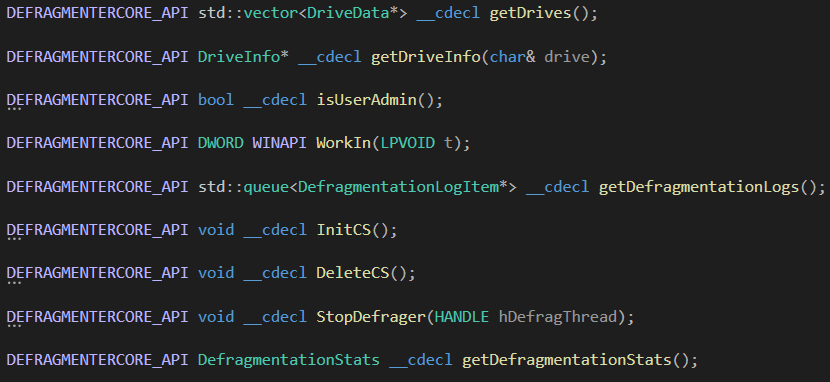


Рисунок 4.1 – API библиотеки DefragmenterCore

Внутри стороннего приложения пользователь может подключить DefragmenterCore так же, как и любую другую DLL-библиотеку: статическим либо динамическим импортом. При статическом импорте необходимо добавить ссылку на библиотеку в зависимости проекта и в целевом файле указать строку #include <путь\_к\_файлу\_Core.h>. Далее можно беспрепятственно вызывать функции библиотеки.

При динамическом импорте с помощью функции LoadLibrary необходимо получить значение hmodule выходного dll файла библиотеки. Далее можно получать указатели на функции библиотеки с помощью вызова GetProcAddress. В него передаётся значение hmodule и строковое представление имени целевой функции. Помимо этого, стоит заметить, что есть возможность получить указатель на саму функцию LoadLibrary внутри библиотеки Kernel32.dll. Для этого используется вызов GetModuleHandle. Подобный пример показан на рисунке 4.2.



Рисунок 4.2 – Получение указателя на LoadLibrary

Несмотря на более высокую сложность использования динамического импорта он является более выгодным с точки зрения расхода памяти. Так, единожды загруженная в оперативную память DLL-библиотека остаётся в ней до самого выключения системы. Она проецируется в виртуальное адресное пространство каждого процесса, который её подключает. Статические библиотеки, напротив, могут существенно увеличивать размер исполняемых файлов за счёт прямой вставки своего содержимого в каждый из них.

В качестве бизнес-логики программного средства для дефрагментации файловой системы NTFS разработанная библиотека может быть использована следующим образом. В заголовочный файл проекта вносится строка #include "../DefragmenterCore/Core.h", а в настройки проекта добавляется ссылка на библиотеку. Далее в обработчике оконной процедуры по сообщению WM\_CREATE инициализируется критическая секция, выбирается буква целевого диска, создаётся поток для работы в нём дефрагментатора и устанавливается периодичность срабатывания таймера для опроса. Пример приведён на рисунке 4.3.

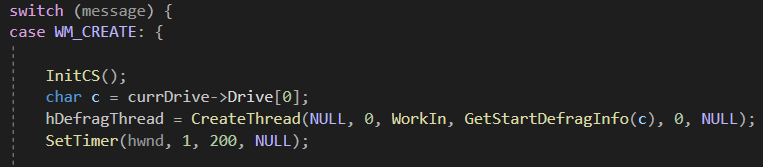


Рисунок 4.3 – Пример организации запуска дефрагментатора

Далее в сообщении WM\_TIMER необходимо получить информацию о процессе дефрагментации с помощью getDefragmentationLogs. Далее данные из полученной очереди возможно отобразить на графический компонент ListView. Пример такого использования отображён на рисунке 4.4.

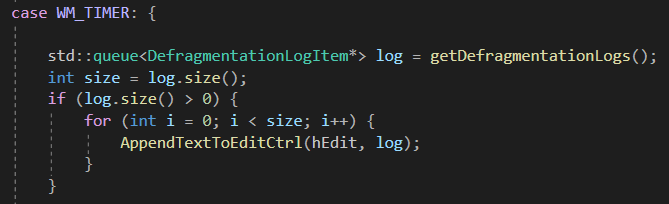


Рисунок 4.4 – Пример сбора данных дефрагментатора

Наконец, в сообщении WM\_DESTROY необходимо остановить процесс дефрагментации, удалить критическую секцию и собрать статистику по обработанным файлам. Пример приведён на рисунке 4.5.

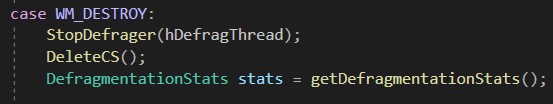


Рисунок 4.5 – Пример остановки дефрагментатора

Готовое приложение, использующее библиотеку DefragmenterCore может отображать информацию о дисковой подсистеме таким образом, как показано на рисунке 4.6.

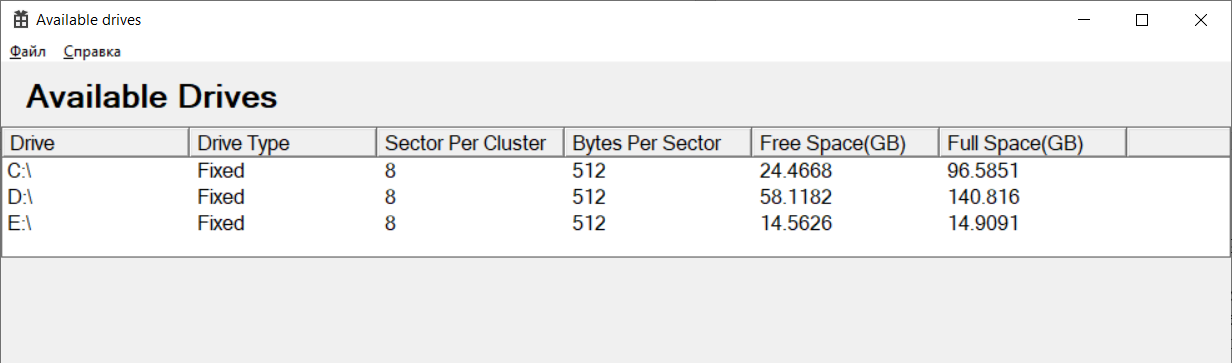


Рисунок 4.6 – Отображение информации о дисках

Полученная с помощью библиотеки информация файловой системы NTFS о конкретном диске может быть отображена пользователю приложения, как показано на рисунке 4.7.

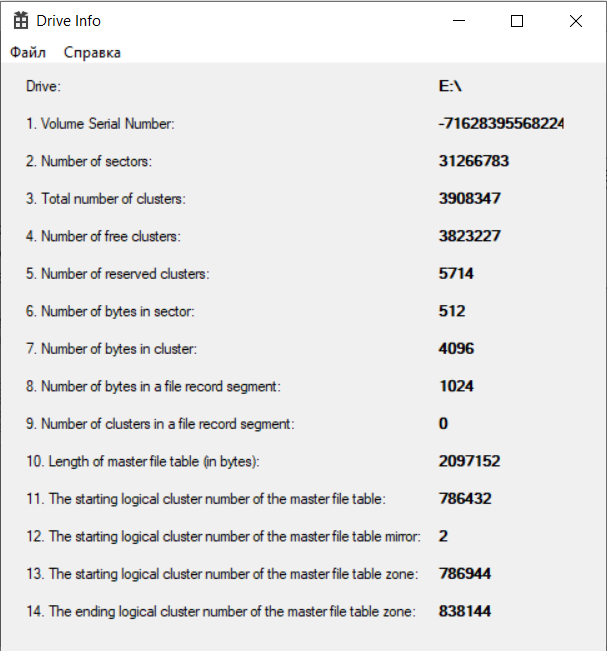


Рисунок 4.7 – Отображение информации NTFS о диске

Для проверки работоспособности дефрагментатора необходимо создать ситуацию фрагментации нескольких файлов на диске. Это может быть достигнуто за счёт увеличения размера файла сверх размера кластера. Тогда оставшиеся данные будут помещены на кластерах в конце дискового пространства, а файл, соответственно, фрагментирован. В примере на рисунке 4.8 таким образом были изменены файлы test.txt и test2.txt. В результате они были дефрагментированы. При следующем запуске дефрагментатора они будут отмечаться со статусом «Clear».

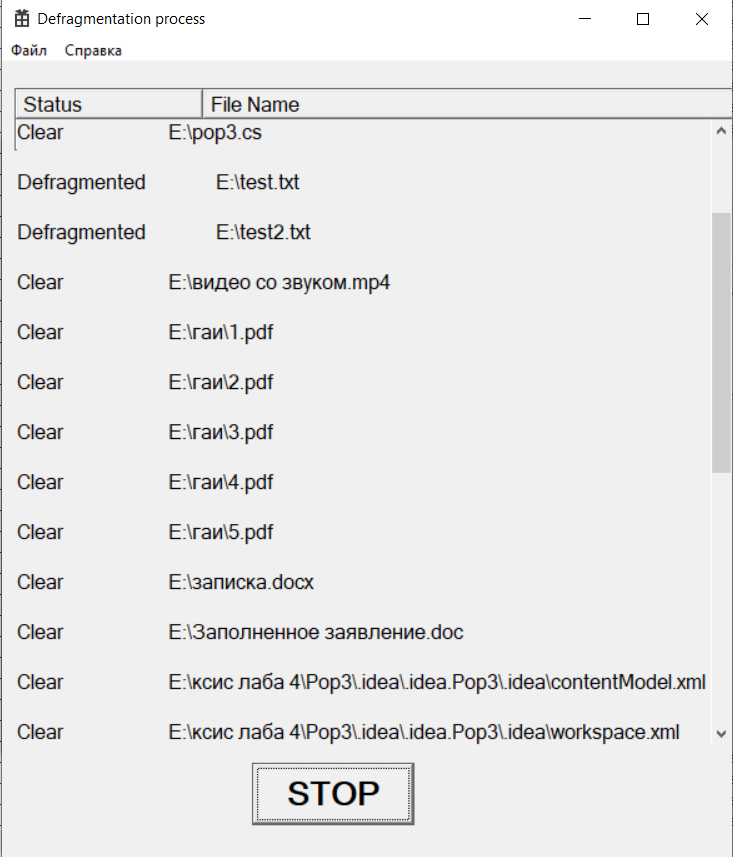


Рисунок 4.8 – Результаты дефрагментации

Следует обратить внимание, что при обычном использовании диска файловая система NTFS сама пытается устранить фрагментацию файлов, для большей скорости чтения. Поэтому ситуацию фрагментации для тестирования необходимо создавать искусственно.

Заключение

В ходе разработки было создана динамически загружаемая библиотека, предоставляющая функции для дефрагментации файловой системы NTFS, а также для сбора информации о дисковой подсистеме. Помимо этого, был реализован функционал для получения информации о процессе дефрагментации, включая статистику по обработанным файлам. В процессе разработки была поставлена и достигнута цель обеспечения безопасной работы библиотеки в многопоточной среде. Предоставлен удобный и понятный API библиотеки, в результате чего была осуществлена её интеграция в графическое приложение, написанное для операционной системы Windows.

Данная DLL-библиотека может быть улучшена, в первую очередь, с помощью пересмотра и оптимизации реализации алгоритма простой дефрагментации. Возможно улучшить алгоритм поиска свободных кластеров на диске, так как в существующем виде он представляет собой простейший линейный поиск. Также стоит реализовать функцию продолжения процесса дефрагментации после его остановки. Для этого необходимо будет зафиксировать последнее состояние дефрагментатора относительно файловой системы. Возможно добавление поддержки файловой системы FAT32.

В ходе написания данного курсового проекта в значительной ступени были улучшены знания системного программирования и операционной системы Windows, а также получены навыки командной разработки. Приобретённые знания будут широко использоваться в дальнейшем.

Список использованной литературы

[1] NTFS [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/NTFS.

[2] NTFS (New Technology File System) [Электронный ресурс]. – Режим доступа: https://ru.bmstu.wiki/NTFS\_(New\_Technology\_File\_System).

[3] Дефрагментация диска [Электронный ресурс] – Режим доступа: https://ru.wikipedia.org/wiki/Дефрагментация\_диска.

[4] Алгоритмы дефрагментации [Электронный ресурс] – Режим доступа: https://studbooks.net/2029180/informatika/opisanie\_algoritmov\_defragmentatsii.

[5] Утилиты дефрагментации Windows [Электронный ресурс]. – Режим доступа: <https://ocomp.info/programmyi-dlya-defragmentatsii.html>.

[6] DeviceIoControl [Электронный ресурс] – Режим доступа: <https://docs.microsoft.com/en-us/windows/win32/api/ioapiset/nf-ioapiset-deviceiocontrol>.

Приложение А

(обязательное)

Исходный код функции дефрагментации файла

int Move(LPCWSTR lpSrcName, LPCWSTR drive)

{

int ret = 0;

ULONG nClusterSize=0, nBlockSize=0, nClCount=0, nFileSize=0, nBytes=0;

HANDLE hDrive = 0, hFile = 0;

ULONG nSecPerCl, nBtPerSec;

MOVE\_FILE\_DATA InBuffer;

ULONG nOutSize = 0;

LPDWORD pBytes = new DWORD[100];

ULONG nCls = 0;

\_\_int64\* pClusters = NULL;

BOOLEAN bResult = FALSE;

LARGE\_INTEGER PrevVCN, LCN;

VOLUME\_BITMAP\_BUFFER\* pOutBuf = readVolumeBitmap(drive);

STARTING\_VCN\_INPUT\_BUFFER InBuf;

RETRIEVAL\_POINTERS\_BUFFER\* OutBuf = NULL;

WCHAR\* Name = new WCHAR[3];

int res = 1;

if (pOutBuf != NULL)

{

res = 0;

hFile = CreateFile(lpSrcName, FILE\_READ\_ATTRIBUTES,

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

NULL, OPEN\_EXISTING, 0, 0);

Name[0] = lpSrcName[0];

Name[1] = ':';

Name[2] = 0;

if (GetDiskFreeSpace((LPWSTR)Name, &nSecPerCl, &nBtPerSec,

NULL, NULL) == FALSE)

{

ret = GetLastError();

delete[] pBytes;

delete[] Name;

delete[] pOutBuf;

CloseHandle(hFile);

return ret;

}

nClusterSize = nSecPerCl \* nBtPerSec;

if (hFile != INVALID\_HANDLE\_VALUE)

{

nFileSize = GetFileSize(hFile, NULL);

nOutSize = sizeof(RETRIEVAL\_POINTERS\_BUFFER) +

(nFileSize / nClusterSize) \* sizeof(OutBuf->Extents);

OutBuf = new RETRIEVAL\_POINTERS\_BUFFER[nOutSize];

InBuf.StartingVcn.QuadPart = 0;

if (DeviceIoControl(hFile, FSCTL\_GET\_RETRIEVAL\_POINTERS,

&InBuf, sizeof(InBuf), OutBuf, nOutSize, pBytes, NULL))

{

nClCount = (nFileSize + nClusterSize - 1) / nClusterSize;

pClusters = new \_int64[nClCount];

PrevVCN = OutBuf->StartingVcn;

for (ULONG i = 0, Cls = 0; i < OutBuf->ExtentCount; i++)

{

LCN = OutBuf->Extents[i].Lcn;

for (int j = (ULONG)(OutBuf->Extents[i].NextVcn.QuadPart

- PrevVCN.QuadPart); j > 0;

j--, nCls++, LCN.QuadPart++)

{

\*(pClusters + Cls) = LCN.QuadPart;

}

PrevVCN = OutBuf->Extents[i].NextVcn;

}

}

}

CloseHandle(hFile);

hFile = CreateFile(lpSrcName, FILE\_READ\_ATTRIBUTES, FILE\_SHARE\_READ |

FILE\_SHARE\_WRITE | FILE\_SHARE\_DELETE, NULL, OPEN\_EXISTING,

FILE\_FLAG\_NO\_BUFFERING, NULL);

hDrive = CreateFile(drive, GENERIC\_READ, FILE\_SHARE\_READ |

FILE\_SHARE\_WRITE, NULL, OPEN\_EXISTING, 0, 0);

if (pClusters)

{

LONGLONG nStartLCN(0), nEmptyCluster(0), nHelpLCN(0),

nMask(1), nInUse(0);

for (\_\_int64 i = 0; i < pOutBuf->BitmapSize.QuadPart; i++)

{

if (pOutBuf->Buffer[i] == 255)

{

nStartLCN += 8;

nEmptyCluster = 0;

continue;

}

while (nMask != 256)

{

if (!nEmptyCluster)

nHelpLCN = nStartLCN;

nInUse = pOutBuf->Buffer[i] & nMask;

if (!nInUse)

nEmptyCluster++;

else

nEmptyCluster = 0;

nMask <<= 1;

nStartLCN++;

}

if (nEmptyCluster >= nClCount)

{

nStartLCN = nHelpLCN;

break;

}

nMask = 1;

}

PrevVCN.QuadPart = 0;

InBuffer.FileHandle = hFile;

InBuffer.StartingLcn.QuadPart = nStartLCN;

for (int k = 0; k < OutBuf->ExtentCount; k++)

{

InBuffer.StartingVcn = PrevVCN;

InBuffer.ClusterCount =

OutBuf->Extents[k].NextVcn.QuadPart–PrevVCN.QuadPart;

if (DeviceIoControl(hDrive, FSCTL\_MOVE\_FILE, &InBuffer,

sizeof(InBuffer), NULL, NULL, &nBytes, NULL) == FALSE)

{

ret = GetLastError();

delete[] pBytes;

delete[] Name;

delete[] pOutBuf;

delete[] OutBuf;

delete[] pClusters;

CloseHandle(hFile);

CloseHandle(hDrive);

return ret;

}

InBuffer.StartingLcn.QuadPart += InBuffer.ClusterCount;

PrevVCN = OutBuf->Extents[k].NextVcn;

}

}

}

delete[] pBytes;

delete[] Name;

delete[] OutBuf;

delete[] pOutBuf;

delete[] pClusters;

CloseHandle(hDrive);

CloseHandle(hFile);

return res;

}

ВЕДОМОСТЬ ДОКУМЕНТОВ

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Обозначение* | | | | | *Наименование* | | | | *Дополнительные сведения* | |
|  | | | | | Текстовые документы | | | |  | |
|  | | | | |  | | | |  | |
| БГУИР КП 1–40 01 01 421 ПЗ | | | | | Пояснительная записка | | | | 32 с. | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | | Графические документы | | | |  | |
|  | | | | |  | | | |  | |
| ГУИР.851004-01 СА | | | | | Дефрагментация файла | | | | Формат А1 | |
|  | | | | | Схема алгоритма | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  |  |  |  |  | БГУИР КП I- 40 01 01 421 ПЗ | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм | Лист | № докум. | Подп. | Дата | Бизнес-логика программного средства для дефрагментации файловой системы NTFS  Ведомость курсового  проекта | Литера | | | Лист | Листов |
| Разраб. | | Пашкевич А.Л. |  | 13.12 | Т |  |  | 32 | 32 |
| Провер. | | Шульга Е.С. |  | 18.12 | Кафедра ПОИТ  гр. 851004 | | | | |
|  | |  |  |  |
|  | |  |  |  |
|  | |  |  |  |