

Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет  
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей  
Кафедра электронных вычислительных машин

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  
к курсовой работе  
на тему  
УТИЛИТА ПОИСКОВ ОДИНАКОВЫХ ФАЙЛОВ  
(аналог fdupes, но с фильтрацией по именам и типам)  
БГУИР КР 1-40 02 01 306 ПЗ

Студент:

Григорик И. А.

Руководитель:

Глоба А. А.

Минск 2022

## Оглавление

ВВЕДЕНИЕ.....	4
1 ОБЗОР ЛИТЕРАТУРЫ.....	6
2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ .....	8
2.1 Постановка задачи.....	8
2.2 Разбиение программы на модули.....	8
2.2.1 Модуль тестирования.....	9
2.2.2 Модуль хранения данных о файлах.....	9
2.2.3 Модуль сбора информации о файлах .....	9
2.2.4 Блок чтения информации о файле .....	9
2.2.5 Блок хеширования данных файла .....	9
2.2.6 Модуль преобразования хеша .....	9
2.2.7 Модуль обработки флагов. ....	10
2.2.8 Блок проверки о хранении файла.....	10
2.2.9 Модуль удаления файлов.....	10
3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ.....	11
3.1 Описание структур утилиты.....	11
3.1.1 file_data_t .....	11
3.1.2 list_t .....	11
3.1.3 flags.....	12
3.2 Описание модулей и блоков программы .....	12
3.2.1 get_dir .....	12
3.2.2 parse_flags .....	13
3.2.3 md5_to_string .....	13
3.2.4 get_size_by_fd.....	13
3.2.5 collect_files.....	13
3.2.6 Блок чтения информации о файле .....	13
3.2.7 Блок получения хеша файла .....	13
3.2.8 Блок проверки о хранении файла.....	14
3.2.9 files_output .....	14
3.2.10 delete_files .....	14
3.2.11 Блок тестового запуска .....	15
3.2.12 generate_files .....	15
3.2.13 test_duplicated .....	15
4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ .....	16
4.1 Выделение ключевых процедур.....	16
4.1.1 Получение длины файла get_size_by_fd.....	16
4.1.2 Вычисление строки хеша файла md5_to_string .....	16
4.1.3 Добавление файла в список add_file_info .....	16
4.1.4 Проверка файла на уникальность check_duplicated .....	16
4.2.1 Удаление файла delete_file.....	17

4.2.2 Удаление всех файлов delete_all_files .....	17
4.2.1 Выбор удаления delete_files.....	18
4.3.1 Генерация случайных файлов generate_files.....	18
4.3.2 Поэтапное тестирование test_duplicated.....	18
5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ.....	19
6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ .....	22
ЗАКЛЮЧЕНИЕ.....	24
ЛИТЕРАТУРА.....	25
ПРИЛОЖЕНИЕ А.....	26
ПРИЛОЖЕНИЕ Б.....	27
ПРИЛОЖЕНИЕ В .....	28
ПРИЛОЖЕНИЕ Г.....	29

## ВВЕДЕНИЕ

В текущее время у каждого пользователя компьютера, вне зависимости от его опыта работы с компьютером, рано или поздно появляются одинаковые файлы. По различным причинам пользователь может не замечать их, или просто игнорировать, т.к. искать вручную все дублирующиеся файлы не так уж и просто. Именно поэтому используются специальные утилиты для автоматического поиска одинаковых файлов.

К примеру, *fdupes* – утилита, написанная Андрианом Лопесом, может искать одинаковые файлы из любого каталога в системе. Для этого используется получение хеша файла из его дескриптора. Следует пояснить, что такое хеш файла и файловый дескриптор:

- *Хеш файла* – это уникальный идентификатор файла, который вычисляется системой посредством определённых преобразований хранящейся в файле информации.
- *Дескриптор файла* – это целое неотрицательное число, с помощью которого процесс может обращаться к потоку ввода-вывода. Дескриптор может быть связан с файлом, сокетом или каталогом.

С помощью данных средств можно сверять файлы по содержанию, что и является целью поиска одинаковых файлов.

Данный курсовой проект представляет собой такую утилиту командной строки, подобную уже известной утилите *fdupes*, которая будет искать файлы с одинаковым содержимым, основываясь на хеше файла. Так же данная утилита будет сравнивать расширения файлов и их имена, помимо содержимого. Так же, с помощью некоторых управляющих флагов, утилита сможет производить действия над данными файлами, помимо простого поиска.

В рамках данной курсовой работы необходимо ознакомиться с библиотекой *openssl/md5.h*, которая используется для получения хеша файлов, или же с командой *MD5*, которые используются для получения *MD5* хеша, с помощью которого файлы и будут проверяться на идентичность. В процессе разработки следует углубить знания по языку *C / C++*, а также осуществить взаимодействие пользовательского приложения с системой. В конце следует протестировать приложение и провести эксперименты на нескольких устройствах.

*OpenSSL* – полноценная криптографическая библиотека, с открытым исходным кодом, которая используется во многих проектах для хеширования *MD5*, *MD2*, *SHA*. Библиотека написана Эриком Янгом и Тимом Хадсоном, и получила популярность благодаря расширениям *SSL/TLS*, которые используются в веб-протоколе *HTTPS*.

Для качественного выполнения курсового проекта следует рассмотреть аналоги данной утилиты. Ключевой аналог – так же консольная утилита *fdupes*. Данная утилита использует хеш файлов, для их сравнения. Утилита написана полностью на языке программирования *C*, и находится под лицензией *MIT*. Она же является примером подражания моего курсового проекта.

Следующая утилита – *CloneSpy*. Это так же бесплатный инструмент для очистки дискового пространства от дублирующихся файлов. Так же может находить файлы нулевой длины, у которых нету содержимого. Данный аналог представляет собой приложение с GUI, что не является целью проекта. Так же приложение разрабатывалось под операционную систему (ОС) Windows, что так же не предпочтительно.

*FSlint* – так же утилита с графическим интерфейсом, но для ОС Linux. Данная утилита представляет собой поиск одинаковых файлов, но с расширенным функционалом. Так же тут присутствуют флаги поиска для одинаковых имён, архивов и пустых директорий.

Рассмотрев все аналоги можно сделать вывод, каковой должна быть программа для корректной конкурентоспособности. Данная утилита должна объединять в себя лучшие качества вышеперечисленных программ, коими были выбраны следующие:

- Обязательный поиск одинаковых файлов из конкретной директории.
- Быстрый поиск файлов, посредством сравнения их хеша.
- Присутствие флагов управления для поиска одинаковых имён.

Данный функционал является минимальным требованием, которое в последующем будет дополнено.

## 1 ОБЗОР ЛИТЕРАТУРЫ

Следует начать с определения утилиты.

*Утилита* – это некая вспомогательная компьютерная программа в составе программного обеспечения, которая используется для выполнения типовых задач, связанной с работой оборудования или ОС. В данном случае утилита используется для облегчения пользования компьютером.

Данная утилита основывается на получении хеша файла из его дескриптора, определения которых давались выше. Процедура получения хеша называется *хешированием*.

Хеширование может проводиться по разным алгоритмам. Основным смыслом хеширования заключается в безопасности и надёжности, возможности сжимать любые куски информации в короткий стандарт сообщений, являющихся уникальными для каждой доли информации. Даже если один байт информации будет изменён – хеш так же поменяется.

Таким образом не надо хранить всё содержимое файла или каждый раз его открывать, чтобы сверить его информацию с другим файлом. Достаточно будет просто сгенерировать его хеш и запомнить его.

Недостатком хеширования является неизбежность коллизии.

*Коллизия* – это равенство значений хеш-функций на двух различных кусках информации. В данном случае это означает, что если функция сгенерирует одинаковый хеш для двух разных файлов, то в системе они будут считаться за одинаковые. Для решения вопроса коллизий создаются современные хеш-функции, в которых шанс появления коллизий стремится к минимуму.

Так же стоит пояснить, что длина строки зависит от конкретной хеш-функции, но одна функция не может сгенерировать две строки разной длины.

На данный момент популярны следующие хеш-функции:

- *SHA256* – одна из наиболее устойчивых к коллизиям функция. Недостаток: по сравнению с другими, имеет довольно большое время выполнения и большая длина хеш-слова (256 байт).
- *RIPEMD160* – так же устойчивая к коллизиям функция, которая, к тому же, имеет длину хеш-слова почти в два раза меньше (160 байт), чем *SHA256*. Время выполнения примерно такое же, как у *SHA256*.
- *MD5* – самая быстрая криптографическая хеш-функция из широко используемых, к тому же имеет наименьший размер хеша (128 байт). Недостаток: небезопасна. Легко подвергается коллизиям, поэтому не стала использоваться в проектах, по типу криптовалютных кошельков.

Так же хеширование используется для сокрытия данных, так как хеширование одностороннее (т.е. нельзя преобразовать хеш в первоначальные данные). Получить первоначальные данные можно только сгенерировав такую же строку, или создать коллизию, которая приведёт к конвертации другой строки

В данном случае сокрытие не требуется, и могут допускаться коллизии, поэтому может использоваться функция хеширования MD5.

Теперь же о файловых дескрипторах.

Файловый дескриптор в данном случае используется для отображения файла на память. Данный метод является эффективным, ибо помогает разгрузить систему, и вообще не использовать физическую память, чем помогает снизить нагрузку на диск для нескольких программ, обращающимся к одному и тому же файлу.

Так же следует заметить, что ОС Linux придерживается правила «всё есть файл», поэтому тип файла тут – понятие, которое отличается и часто путается с расширением файла в ОС Windows. В Linux существуют всего три типа файлов: обыкновенные, специальные и директории. Следовательно, можно сделать вывод, что условие реализации фильтрации поиска по типам файла будет реализовываться относительно их расширения, что является подтипом обыкновенных файлов.

## **2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ**

Сначала следует поставить конкретные функциональные требования разрабатываемой программе, разбить утилиту на модули и функциональные блоки. Данный подход в большинстве упростит понимание проектирования, сможет помочь устранить проблемы в архитектуре и обеспечить гибкость каждого из модулей. После того, как аналоги рассмотрены и проведен краткий экскурс в основные понятия, можно поставить конкретные цели разрабатываемому программному обеспечению.

### **2.1 Постановка задачи**

Минимальное требование – просто поиск одинаковых файлов с определённой директории, с из выводом на экран. Так как в условии курсового проекта сказано, что должна быть фильтрация по типам, то это так же будет являться обязательным условием выполнения. В качестве поиска будет использоваться алгоритм хеширования MD5 из-за своей скорости и маленькой длине хешируемого слова, что является условием быстрого выполнения утилиты. Так же для оптимизации будут использоваться отображения в память, для упрощённого, для системы, получения хеша файла. Данный подход поможет не затрачивать лишнее дисковое пространство и в несколько раз ускорить работу утилиты. Так как присутствует условие фильтрации по типам, то утилита должна подстраиваться под передаваемые ей пользовательские флаги, следовательно, должен быть установлен обработчик входных флаговых значений.

Из вышеперечисленных методов можно составить определённые модули программы, которые будут обеспечивать полную функциональность выполнения. Так же следует выделить, что в данном случае расширение файла, и его имя является одним и тем же, следовательно, для этого не имеет смысла делать двух разных флагов, и будет организован только один флаг: `-n (name)`. Так же к флагам добавляется флаг статистики (`-s`), и флаг удаления файлов (`-d`). Первый будет использоваться для показа текущей статистики сбора файлов, второй для полного удаления дубликатов файлов после их поиска. Дополнительный флаг – флаг сбора всех файлов `-a (all)`, используется в различных утилитах, для поиска системных файлов, или скрытых файлов, начинающихся с точки. Последний флаг – флаг примера (`-t`), который будет отображать корректное поведение программы. С него и стоит начать описание модулей программы.

### **2.2 Разбиение программы на модули**

Для корректного решения излагаемой проблемы всю программу следует разбить на модули, которые впоследствии будут реализованы в функциях или блоках кода. Модулем будет являться полноценная функция или подфункция, а блоком – блок кода, который может содержаться в модуле.



### **2.2.1 Модуль тестирования**

Данный модуль будет полностью симулировать применение программы. Он будет представлять собой открытие какого-то каталога, создания там нескольких файлов с одинаковым содержимым, выводом этого содержимого на экран, задержкой для проверки содержимого и удалением или поиском одинаковых файлов. Модуль необходим для отображения корректного поведения программы, и будет использовать максимальное количество модулей, описанных ниже.

### **2.2.2 Модуль хранения данных о файлах**

Данный модуль будет представлять собой некий массив самостоятельно написанной структуры, которая будет использоваться для хранения всех метаданных о каждом проверяемом файле. В данном случае метаданными будут служить хеш файла и его имя, для уникальных файлов, и имя файла и путь до него, для дублирующихся файлов. Данный модуль будет представлять из себя два вектора данных структур.

### **2.2.3 Модуль сбора информации о файлах**

Модуль блок будет представлять собой рекурсивную функцию, используемую для прохождения по всем файлам с задающей директории. Модуль необходим для обновления информации модуля хранения данных о файлах. В нём же можно выделить ещё несколько блоков и модулей, которые будут описаны в заголовках **2.2.5**, **2.2.6** и **2.2.7**.

### **2.2.4 Блок чтения информации о файле**

Блок будет реализовывать собой отображение данных файла в виртуальное адресное пространство, после чего будут задействован блок получения хеша файла и модуль преобразования хеша в шестнадцатеричную систему счисления (СС).

### **2.2.5 Блок хеширования данных файла**

Данный блок кода просто будет представлять собой получения хеша определённого файла основываясь не его отображении в виртуальном адресном пространстве. После выполнения этого блока необходимо освободить адресное пространство.

### **2.2.6 Модуль преобразования хеша**

Данный модуль преобразовывает полученный хеш, который может состоять из нечитаемых символов или же из символов, неудобных для работы, в

строку, определённой длины, состоящую из шестнадцатеричных чисел. Преобразование необходимо для корректной работы, ибо данные в неудобных форматах для чтения могут неправильно сравниваться.

### **2.2.7 Модуль обработки флагов.**

Модуль представляет собой преобразование входных данных в определённые флаги, которые в дальнейшем будут использоваться в блоке проверки данных и выводе.

### **2.2.8 Блок проверки о хранении файла.**

Блок является проверкой данных конкретного файла со всеми файлами, которые хранятся в модуле хранения данных. Модуль может изменяться, в связи с изменением некоторых флагов. В случае выполнения определённых условий, данные файла будут заноситься в вектор дублирующихся файлов.

### **2.2.9 Модуль удаления файлов**

Модуль является простым считыванием вектора дублирующихся файлов и их последующего удаления. Так же возможно не полное, а выборочное удаление файлов.

Все вышеперечисленные модули позволяют обеспечить полноту выполняемых действий, соответственно необходимы для выполнения курсовой работы.

### 3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Данная глава будет представлять собой ключевой раздел, дающий понимание работы моей утилиты, её структуру, с точки зрения описания отдельных функций и модулей, обработки данных, с приведением листинга и структурной диаграммой, вынесенной в приложение «А».

#### 3.1 Описание структур утилиты

Ключевыми структурами утилиты являются `file_data`, `list` и `flags`.

Первые две структуры представляют собой шаблоны для хранения имени файла и его хеша и пути до него и объединение данных в связанный список. Первая структура является удобным объединением данных о конкретном файле, а вторая является реализацией этого объединения в список. Данные структуры являются ключевыми в поиске и удалении файлов, тем не менее их можно рассматривать и по-отдельности.

##### 3.1.1 `file_data_t`

Структура представляет собой всего два поля – сгруппированные данные о файле, из которых можно определить его уникальность. Поля:

- `file_name` – строка, которая хранит имя файла с его относительным путём.
- `file_hash` – строка, хранящая хеш файла в шестнадцатеричном коде, с помощью которой и определяется уникальность файла.
- `path` – строка, содержащая абсолютный путь до файла.

С помощью всего лишь двух полей (`file_hash` и `file_path`) можно реализовать систему поиска одинаковых файлов, но я решил добавить имя файла исключительно для удобства использования. Но если бы дублирующиеся и уникальные файлы никуда не записывались, то смысл этой структуры и утилиты в целом теряется, так что для полного функционала необходима следующая структура:

##### 3.1.2 `list_t`

Данная структура является минимальным требованием, для реализации сбора уникальных и дублирующихся файлов в системе. Она содержит поле информации о файле и указатель на следующий объект. Поля:

- `file_data` – поле, используемое для хранения данных о файле. Является типом структуры `file_data_t`.
- `*next` – указатель на следующий объект данной структуры.
- `*tail` – указатель на хвост структуры (необходим для быстрого добавления данных в конец).

Кроме этого в данном проекте представлена система управления программой посредством флагов, поэтому необходимость структуры, содержащей флаги в виде булевых переменных крайне важна.

### 3.1.3 flags

Структура flags будет реализовывать собой простой набор булевых переменных, которые будут выставляться, в зависимости от ввода пользователя. Флаги же могут выставляться как отдельно, так и совместно. Это будет описано в полях:

- `bool stats` – вывод статистики по собираемым файлам как в момент сборки и поиска, так и в момент конечного вывода.
- `bool name_flag` – флаг фильтрации по именам. Позволяет управлять сборкой информации о файлах.
- `bool delete_flag` – флаг для удаления повторяющихся файлов. Активирует модуль удаления, который без выставления данного флага будет являться недействительным.
- `bool all_files` – флаг, позволяющий собирать информацию в скрытых файлах, так же известных как dot-файлы в Linux.
- `bool test_flag` – данный флаг позволяет полностью симулировать работу программы. Указываемый путь после этого файла не будет использоваться и не обязателен.

Стоит отметить, что все вышеперечисленные флаги могут совмещаться, но отдельный флаг `test_flag` будет игнорировать все поставленные флаги, т.к. после его установки пользователь ожидает увидеть полную работу программы со всеми доступными флагами по очереди. Чтобы сделать это возможным программа последовательно выставляет все флаги (по одному), и запускает модули поиска дублирующихся файлов, который в себе вызывает остальные блоки и подмодули. Данные модули будут описываться в следующем разделе.

## 3.2 Описание модулей и блоков программы

Все вышеописанные структуры являются полностью бесполезными, без средств работы с ними, так называемыми «модулями», или функциями программы, которые содержат блоки кода. Рассматривать их следует от самого простого к самому сложному:

### 3.2.1 get\_dir

Данный модуль один из самых простых, и позволяет преобразовать строку, введенную пользователем одной из аргументов командной строки, в правильный формат. К примеру, если введена директория с относительным путём `/home/`, то она преобразуется в строку `/home`. Этот модуль необходим для корректного открытия директории и вывода пути до файла в будущем.

### **3.2.2 parse\_flags**

Модуль представляет собой функцию, возвращающую структуру типа `file`. Получает аргументы командной строки и их количество. Создаётся новый объект структуры `file_data`, заполняется на основании содержания аргументов командной строки и возвращается.

### **3.2.3 md5\_to\_stirng**

Функция представляет собой конвертацию не всегда читаемых символов в строку определённой длины (длины хеш-слова `md5`). Строка конвертируется посимвольно путём преобразования каждого символа в шестнадцатеричную СС. В конце конвертации эта строка возвращается.

### **3.2.4 get\_size\_by\_fd**

Функция получает размер файла, на основе его дескриптора, создавая переменную типа `struct stat` и вызывая функции `fstat`, находящиеся в библиотеке `sys/types.h`, `sys/stats.h`, `unistd.h`. Возвращает значение поля `st_size` структуры `stat`, в удачном случае, и завершает программу с кодом `-1` в неудачном.

### **3.2.5 collect\_files**

Этот модуль является одним из ключевых. Представлен в виде функции, которая может вызываться рекурсивно, и получает на вход ссылки на указатели структур `list_t` в трёх экземплярах, директорию входа и переменную типа `flags`, которые используются для сбора информации и её фильтрации. В целом структура модуля разбивается на вышеописанные (3.2.(3-4)), и внутренние условия. Изначально модуль создаёт некоторую переменную `dir` – указатель на директорию, в которой работает программу. Далее создаётся поток каталога, в котором работает программа, и пока он не будет полностью считан – не произойдёт выход из функции. Этот цикл и называется модулем сбора информации, внутри которого и выполняются все блоки программы (2.2.4 – 2.2.8). Данные блоки будут описываться ниже:

### **3.2.6 Блок чтения информации о файле**

Если файл существует и выполняются все условия (файл не «..» или «.» и не директория), то программа будет читать файловый дескриптор, и получать посредством вызова `get_size_by_fd` длину файла и в случае, если она меньше 1 Гб отображает файл в память и активирует следующий блок:

### **3.2.7 Блок получения хеша файла**

Блок представляет собой вызов функции `MD5` из библиотеки `openssl/md5.h`, которая заполняет строку `result` – строку для записи хеша, на

основе отображения файла в памяти и его размера. После успешного хеширования удаляет отображение файла из виртуальных адресов и конвертирует хеш в строку, с помощью модуля `md5_to_string`. После этого выводится статистика, если подобный флаг был выставлен и обработан в `parse_flags`, и функция приступает к главному: блоку проверки о хранении файла.

### **3.2.8 Блок проверки о хранении файла**

Данный блок является циклом, проходящим по каждому элементу структуры `list **unique_files` со структурой `file_data`, что позволяет сравнить конкретный файл со всеми хранящимися в массиве. Блок сверяет хеш текущего файла и хранимого. В случае эквивалентности, информация о файле заносится в `list **files_to_delete`. Если же все файлы проверены, то информация заносится в массив `unique_files`. Если же в массиве ничего не хранится – то файл записывается, и программа переходит к другому файлу. После этого блока функция заканчивается, и программа выполняет следующий цикл рекурсии или переходит к следующим модулям.

### **3.2.9 files\_output**

Модуль представляет собой функцию, которая выводит содержимое переданных в него структур типа `list_t`. Если какой-либо из указателей на структуру пустой (т.е. нету повторяющихся файлов или просто нету файлов), то выводится сообщение о отсутствии файлов или дублирующихся файлов. Если флаг вывода статистики активирован, то функция выводит дополнительную статистику о количестве дублирующихся и уникальных файлов. И переходит к следующему, заключительному модулю.

### **3.2.10 delete\_files**

Если был выставлен флаг удаления, то этот модуль активируется. В противном случае программа завершается. Модуль является функцией, принимающей массив файлов для удаления и флаги, выставленные в самом начале программы. Функция начинается информационным сообщением и вопросом, какой конкретно файл надо удалить, удалить все файлы или выйти. Если вводится число – то удаляется дублирующийся файл, который пронумерован данным числом. Если такого числа нету – пользователю выводится сообщение об ошибке диапазона. В случае ввода буквы «А» (All) – удаляются все файлы из списка. В случае ввода цифры «0» - модуль завершает работу после вывода статистики, если она не выводилась до этого. После выполнения данного модуля программа завершает выполнение, если не выполняется тестовый запуск, описание которого будет изложено ниже.

### **3.2.11 Блок тестового запуска**

Данный блок является полным тестированием всей программы, который выполняется постепенно с ожиданием ввода от пользователя, который будет означать переход к следующему шагу тестирования. Блок позволяет тестировать всю программу за счёт внутренних подмодулей.

### **3.2.12 generate\_files**

Данный подмодуль является функцией генерации файлов с псевдослучайным содержимым. Все файлы генерируются в отдельной папке, идущей вместе с проектом. Наполнение файла состоит из 3 символов, каждый из которых может быть или 0 или 1. После открытия файла для записи, происходит сама генерация символов, каждого по-отдельности. После генерации в консоль выводится имя файла и строка из трёх сгенерированных символов, которые будут туда записаны. Данная генерация позволяет каждый раз получать новые значения содержимого файлов, после чего искать среди них уникальное наполнение. После генерации всех 9 файлов программа запускает следующий модуль – модуль показа поиска

### **3.2.13 test\_duplicated**

Модуль иллюстрирует поиск файлов с разными флагами. Является функцией, которая создаёт указатель на структуру с уникальными и одинаковыми файлами для записи значений в них и переменной флагов, которая в последующем будет изменяться для симуляции поиска. Данная функция меняет значения флагов этой переменной и вызывает модуль показа содержимого файлов и поиска одинаковых файлов с этими флагами. В конце происходит показ удаления файлов с флагами по умолчанию (или же их отсутствием). Таким образом модуль затрагивает каждый флаг и показывает, как он работает.

Данные модули и блоки реализуют основную логику программы, и являются зависимыми друг от друга по древовидной структуре, поэтому в данной утилите довольно сложно удалить один сегмент кода, не повливав на другой.

## **4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ**

### **4.1 Выделение ключевых процедур**

Ключевыми процедурами утилиты для поиска одинаковых файлов можно назвать процедуру получения длины файла, вычисления строки хеша файла, добавления файла в список и проверки файла на уникальность. Для удаления файла из списка нужны следующие процедуры: выбора удаления, удаления файла, и процедура удаления всех файлов. Для тестирования важны все вышеперечисленные процедуры, с добавлением следующих процедур: генерация случайных файлов и процедура поэтапного тестирования.

Все процедуры будут описаны в вышеперечисленном порядке.

#### **4.1.1 Получение длины файла `get_size_by_fd`**

Данная процедура должна принимать дескриптор файла `fd`. Внутри создаётся переменная `statbuf` типа `struct stat`, для дальнейшего заполнения путём использования функции `fstat`. Далее проверяется правильность выполнения функции путём сравнения с нулём и если проверка пройдена – возвращается значение `statbuf.st_size`, которое и является длиной файла.

#### **4.1.2 Вычисление строки хеша файла `md5_to_string`**

Принимает указатель типа `unsigned char` (данный тип возвращает функция MD5). Внутри создаёт переменную типа `char*` - `hash`. В цикле конвертирует каждый символ из принятого массива в шестнадцатеричную систему счисления путём выполнения функции `sprint` и записывает его в переменную `hash`. В конце возвращает данный указатель на переменную.

#### **4.1.3 Добавление файла в список `add_file_info`**

Функция принимает на вход структуру `list_t **to_add`, в которую следует добавить информацию, имя файла, путь до него и его хеш (`char *filename`, `char *file_path`, `char *hash`). Далее создаётся временная переменная типа `list_t *temp`, в которую копируется хеш файла, путь до него и его имя. Указатель на следующий объект структуры ставится в `NULL`. Далее проверяется, есть ли что-либо в списке, и если нету – то указатель на начало списка ставится на переменную `temp`. Хвост списка ставится в его начало. Если же в списке что-либо есть, то указатель на следующий элемент после хвоста ставится на `temp` и хвост передвигается на следующий элемент

#### **4.1.4 Проверка файла на уникальность `check_duplicated`**

Данная процедура является ключевой во всей курсовой работе. На вход принимает `char *filename`, `char *file_path`, `char *hash`, структуры типа `list_t **unique_files`, `duplicated_files` для добавления в них информации и флаги типа



flags\_t для фильтрации обрабатываемой информации. Изначально проверяется, есть ли в структуре unique\_files какие-то данные ((\*unique\_files) != NULL). Если никаких данных нету – то в неё заносятся данные о первом файле, и функция оканчивается. Если же в ней что-либо есть – то создаётся указатель list\_t \*ptr, который указывает на начало структуры данных (в будущем служит для обхода каждого элемента структуры). Далее в бесконечном цикле сравнивается хеш файла, переданного в функцию и хеш файла, записанного в текущем элементе. Если они совпадают – то если выставлен флаг фильтрации по именам – сравниваются имена и если совпадает – то заносится в duplicated\_files, а если не совпадает – то цикл продолжается. Если же флаг фильтрации по именам не выставлен – то информация о файле заносится в duplicated\_files, и выходим из бесконечного цикла. Далее проверяется, есть ли в списке следующий элемент, и если есть, то ptr принимает значение следующего элемента. Если же нету – то выходим из цикла и добавляем информацию в структуру unique\_files и выходим из функции.

Данный блок является трудным в понимании, поэтому в приложении А будет приведён его листинг.

#### **4.2.1 Удаление файла delete\_file**

Процедура принимает структуру из файлов list\_t \*\*duplicated\_files и номер файла, который следует удалить. Создаются две временные переменные list\_t \*temp, равная началу списка и \*prev. Если номер файла, который надо удалить, равен 1, и структура не является пустой, то файл удаляется путём вызова функции remove, начало списка сдвигается на следующее значение, предыдущее значение начала списка, записанное в переменную temp, очищается и функция оканчивается. Если же номер больше единицы, то в цикле присваивается значение переменной prev переменной temp и переменная temp ставится на temp->next, пока номер итерации не будет равен номеру выбранного файла. Далее если переменная temp равна NULL (если такого файла не существует), то мы возвращаемся из функции. Если же такой файл существует, то мы его удаляем и ставим указатель prev->next на temp->next, после чего очищаем переменную temp и выходим из функции.

#### **4.2.2 Удаление всех файлов delete\_all\_files**

В функцию удаления всех файлов передаётся указатель на структуру duplicated\_files. Функция начинается с предупреждения, что удаление дубликатов с корневой директории или с домашней директории могут привести к непоправимым последствиям. Если же пользователь согласен (если он ввёл Y), то ставится указатель на начало списка, и программа входит в цикл, в котором поэтапно удаляет файл и перемещается на следующий элемент в списке.

### 4.2.1 Выбор удаления `delete_files`

Функция принимает на вход только структуру `list_t **duplicated_files`. После идёт проверка, если структура `*duplicated_files` пуста (т.е. дубликатов нету), то выводится сообщение об отсутствии дубликатов, и функция завершается. Если же не пуста, то в цикле выводятся пронумерованные дубликаты файлов и пользователю требуется ввести номер файла, чтобы удалить конкретный файл, 0 – чтобы удалить все файлы, или N – чтобы закончить выполнение программы. Если пользователь вводит N – то выводится сообщение о завершении, и программа заканчивается. Если пользователь вводит 0 – то активируется блок `delete_all_files`, и программа завершается. Если вводится цифра – то активируется блок `delete_file` и цикл выполняется заново, если не удалены все файлы.

### 4.3.1 Генерация случайных файлов `generate_files`

Данная функция генерирует файлы для последующего тестирования. Использует библиотечную функцию `rand`. Создаёт переменную `char fil_name[]`, которая хранит абсолютный путь до файла и переменную `char content[4]`, которая хранит содержимое файла. В цикле последовательно к последнему символу добавляется новая цифра (от 1 до 9), файл открывается для записи (или же создаётся), после чего во внутреннем цикле генерируется наполнение файла. Каждый символ строки `content [1...3]` заполняется случайным значением (или 0 или 1) посредством вызова функции  $(\text{rand}() \% 2) + 48$ . После генерации содержимое записывается в файл, файл закрывается и переходим к другому файлу.

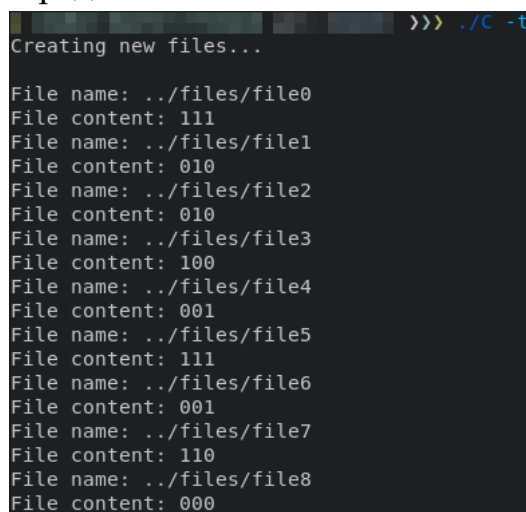
### 4.3.2 Поэтапное тестирование `test_duplicated`

В данной функции создаётся переменная `char dir[]`, в которую записывается директория где будут случайно сгенерированы файлы, переменная типа `flags_t`, для дальнейшего тестирования поиска с различными флагами, и указатели на списки `list_t *unique_files`, `*duplicated_files`, `*error_files`, в которые будут записаны уникальные, дублирующиеся файлы и файлы, которые невозможно было обработать. Далее вызывается функция поиска дублирующихся файлов и вывод статистики, в котором происходит всё, описанное в пунктах 4.1.(1-4), и вывод статистики обновлённых структур.

Так же в программа не может функционировать без алгоритмов перебора файла и открытия директорий. Эти алгоритмы будут вынесены в блок-схемы.

## 5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

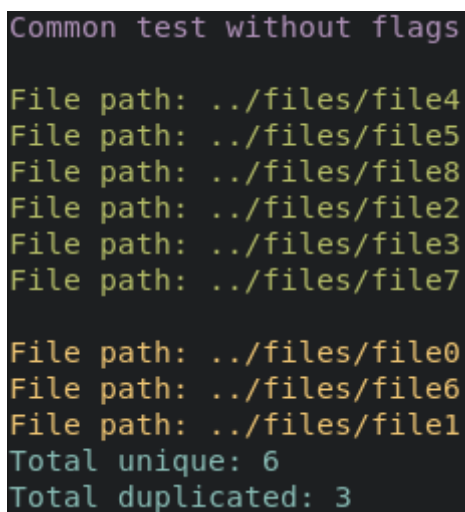
В данной программе существуют внутренние методы тестирования. Они вызываются, когда программа запускается с флагом `-t` (test). Метод заключается в поэтапной генерации 9 файлов со случайным наполнением и в последующем вызове функций нахождения дубликатов и удаления дубликатов. Утилита поставляется вместе с папкой `files`, в которой и создаются файлы для дальнейшего тестирования. Всего выполняется два этапа. Первый этап – запуск обычной проверки утилиты без флагов. Второй этап – запуск утилиты с флагом удаления. Таким образом после вызова утилиты с данным флагом все дублирующиеся файлы в папке `files` будут удалены. Работоспособность программы с данным флагом представлена ниже.



```
>>> ./c -t
Creating new files...
File name: ../files/file0
File content: 111
File name: ../files/file1
File content: 010
File name: ../files/file2
File content: 010
File name: ../files/file3
File content: 100
File name: ../files/file4
File content: 001
File name: ../files/file5
File content: 111
File name: ../files/file6
File content: 001
File name: ../files/file7
File content: 110
File name: ../files/file8
File content: 000
```

Рисунок 5.1 – генерация файлов со случайным наполнением

После данной генерации можно заметить, что файлы 0 и 5, 1 и 2, 4 и 6 совпадают, следовательно, какой-либо файлов в этих парах должны быть занесены в список дублирующихся файлов.



```
Common test without flags

File path: ../files/file4
File path: ../files/file5
File path: ../files/file8
File path: ../files/file2
File path: ../files/file3
File path: ../files/file7

File path: ../files/file0
File path: ../files/file6
File path: ../files/file1
Total unique: 6
Total duplicated: 3
```

Рисунок 5.2 – Проверка уникальности без флагов.

После проверки файлов на уникальность видно, что файлы 0, 1 и 6 были занесены в список дублирующихся файлов, а файлы 5, 2 и 4 – в список уникальных файлов.

Следующий этап представляет собой поиск дубликатов тех же файлов с флагом удаления. Так как в программе ничего не менялось, то результат поиска должен соответствовать предыдущему результату. После поиска удаляется первый найденный файл (файл 5).

```
Test with deletion flags

File path: ../files/file4
File path: ../files/file5
File path: ../files/file8
File path: ../files/file2
File path: ../files/file3
File path: ../files/file7

File path: ../files/file0
File path: ../files/file6
File path: ../files/file1
Total unique: 6
Total duplicated: 3
```

Рисунок 5.2 – Поиск дубликатов файлов с флагом удаления.

Как видно, ничего не изменилось, следовательно, можно перейти к удалению файлов.

```
Delete 1st file

File path: ../files/file4
File path: ../files/file5
File path: ../files/file8
File path: ../files/file2
File path: ../files/file3
File path: ../files/file7

File path: ../files/file6
File path: ../files/file1
Total unique: 6
Total duplicated: 2
```

Рисунок 5.3 – Удаление первого файла из списка и из системы.

После этого можно проверить директорию и убедиться, что удалённого файла там больше нету.

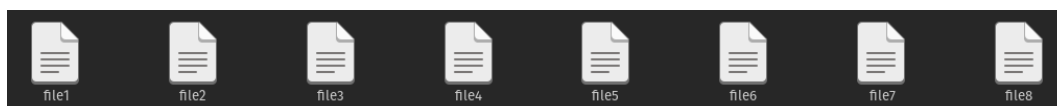


Рисунок 5.4 – Директория после удаления файла 0.

Как видно, удалённого файла нету, следовательно, программа отработала правильно.

Так же следует протестировать программу с выставлением флага фильтрации имени и без. Если флаг фильтрации имени выставлен, то два файла с одинаковым наполнением и разным именем могут находиться в списке уникальных файлов. Проведём тестирование сначала без флага уникальных флагов:

```

>>> ./C /home/sifi/DOS/check_files/
File path: /home/sifi/DOS/check_files/test/5lab2.txt
File path: /home/sifi/DOS/check_files/test/file.txt 1
File path: /home/sifi/DOS/check_files/test/5lab1.txt
File path: /home/sifi/DOS/check_files/text.txt 2
File path: /home/sifi/DOS/check_files/text (copy).txt 2
File path: /home/sifi/DOS/check_files/file.txt 1
File path: /home/sifi/DOS/check_files/file (copy).txt 1
Total unique: 4
Total duplicated: 3
>>>

```

Рисунок 5.5 – Поиск дубликатов без флагов

Как видно, файлы, помеченные цифрой 1 и 2 одинаковые, но второй файл в первом списке и второй файл во втором списке имеют одинаковые имена. Теперь, когда выставлен флаг фильтрации, списки будут выглядеть так:

```

>>> ./C -n ~/DOS/check_files/
File path: /home/sifi/DOS/check_files/test/5lab2.txt
File path: /home/sifi/DOS/check_files/test/file.txt
File path: /home/sifi/DOS/check_files/test/5lab1.txt
File path: /home/sifi/DOS/check_files/text.txt
File path: /home/sifi/DOS/check_files/text (copy).txt
File path: /home/sifi/DOS/check_files/file (copy).txt
File path: /home/sifi/DOS/check_files/file.txt
Total unique: 6
Total duplicated: 1
>>>

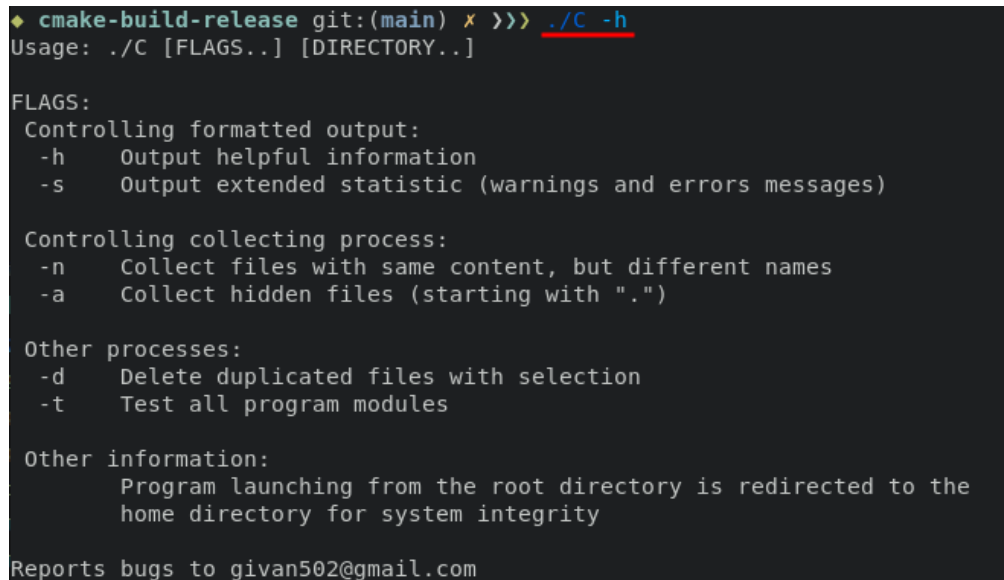
```

Рисунок 5.6 – Поиск дубликатов с фильтрацией по имени

Теперь видно, что в список дублирующихся файлов был занесён файл с таким же именем, а все остальные были убраны. Следовательно, работоспособность программы протестирована.

## 6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

В данной утилите сложно запутаться из-за того, что её использование полностью сводится к написанию одной программы и выставлению определённых флагов, но всё же была написана функция – помощник, которая выводит информацию об использовании утилиты. Данная функция активируется, если в команду запуска были переданы неверные аргументы или если программа запущена с флагом `-h` (`--help`).



```
◆ cmake-build-release git:(main) x >>> ./C -h
Usage: ./C [FLAGS..] [DIRECTORY..]

FLAGS:
Controlling formatted output:
-h    Output helpful information
-s    Output extended statistic (warnings and errors messages)

Controlling collecting process:
-n    Collect files with same content, but different names
-a    Collect hidden files (starting with ".")

Other processes:
-d    Delete duplicated files with selection
-t    Test all program modules

Other information:
      Program launching from the root directory is redirected to the
      home directory for system integrity

Reports bugs to givan502@gmail.com
```

Рисунок 6.1 – Вывод программы при активации флага помощи.

Данная страница полностью показывает, что следует передавать в аргументы программы. Т.е. если необходимо запустить программу с флагами вывода расширенной статистики с домашней директории, то необходимо запустить программу так: «`./C -s /home`».

Так же флаги можно передавать путём конкатенации строки, начинающейся с символа «`-`». К примеру, если надо выставить флаги удаление и сбора скрытых файлов, то можно написать, как `-d -a`, так и `-ad`.

Стоит упомянуть, что при запуске программы с флагом тестирования все остальные флаги игнорируются. Так же при запуске программы с флагом помощи, все остальные флаги так же будут игнорироваться. Т.е. сверху иерархии флагов стоит флаг тестирования, за ним флаг помощи, и потом все остальные флаги.

Как отмечено в странице-мануале, запуск программы с корневой директории должен быть перенаправлен с домашнюю директорию пользователя, запускавшего программу, так как данная утилита должна быть доступна для использования на ssh-серверах и локальных серверах. В случае, если на таковых

она будет запущена с домашней директории, то файлы других пользователей или самой системы могут быть повреждены.

Так же это является бесплатной утилитой с открытым исходным кодом, и кто угодно может информировать о её недостатках или предложить вносить свои изменения. Для этого можно написать на почту [givan502@gmail.com](mailto:givan502@gmail.com) или же послать запрос на изменение в сам репозиторий с исходным кодом утилиты, который находится по адресу [github.com/IvanGrigorik/MF](https://github.com/IvanGrigorik/MF).

## ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были приобретены и укреплены знания в сфере системного программирования, такие как: отображение файлов в память, POSIX-совместимые файловые системы, получение информации о файле по его дескриптору, хеширование файлов и многие другие. Благодаря вышеописанным знаниям удалось создать полностью рабочую утилиту, основанную на получении хеша файлов и его сравнения. Так же удалось оптимизировать процесс сбора файлов, путём создания очереди, в которой элемент может вставляться сразу в хвост, путём отображения файла в память для получения его дальнейшего хеша. Стоит упомянуть, что углубившись в знания хеширования удалось понять, какой алгоритм тут является самым эффективным и почему.

Выполненная программа не только полностью обеспечивает выполнение заданных требований, но и имеет свои достоинства, к примеру интерактивное тестирование.

Из достоинств следует выделить скорость выполнения программы (0.2 секунды по сравнению с 0.44 с программой, написанной на C++), полную функциональность и стабильное выполнение.

Из недостатков можно выделить возможное получение одинакового хеша для разных файлов и невозможность сканирования файлов, размером свыше 1 Гб. Вероятность возникновения первого крайне мала и решается переписыванием получения и сравнения хеша на другой алгоритм хеширования, а второе является встроенным недостатком функции `mtar`.

Путей совершенствования приложения довольно много, поэтому код будет находиться в открытом доступе в моём репозитории. При желании можно отправить мне запрос на совместное использование или расширение функционала программы.

Данный курсовой проект помог понять, как устроены многие приложения с командным интерфейсом, показать их недостатки и понять, как их можно усовершенствовать.



## ЛИТЕРАТУРА

1. Лафоре, Р. Объектно-ориентированное программирование в C++ / Р. Лафоре. – СПб. : Питер, 2004.
2. Буч, Г. Язык UML. Руководство пользователя / Г. Буч, Д. Рамбо, А. Джекобсон; пер. с англ. – СПб. : ДМК, 2004.
3. Лав Р. Системное программирование на Linux/ 2-е издание 2014.
4. Керниган Б. Язык программирования C/ 4-е издание М.:Питер, 2004. – 923 с.
5. Рочкинд М. Программирование для UNIX, 2-е изд. СПб, БХВ-Петербург, 2005.
6. Калле Р. Грокаем технологию биткойн / Р. Калле – СПб. : Питер, 2020.

## **ПРИЛОЖЕНИЕ А**

*(Обязательное)*

Листинг модуля проверки файла на уникальность с комментариями

## **ПРИЛОЖЕНИЕ Б**

*(обязательное)*

Блок-схема алгоритма перебора файлов

## **ПРИЛОЖЕНИЕ В**

*(обязательное)*

Блок-схема алгоритма открытия директорий

## **ПРИЛОЖЕНИЕ Г**

*(обязательное)*

Схема функциональных блоков программы