Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Операционные системы и системное программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

УТИЛИТА КОНТРОЛЯ ПОЯВЛЕНИЯ ДУБЛИКАТОВ В ФАЙЛОВОЙ СИСТЕМЕ С ЗАМЕНОЙ ИХ НА ЖЕСТКИЕ ССЫЛКИ И ПРОТОКОЛИРОВАНИЯ ФАКТОВ ЗАМЕНЫ

БГУИР КП 1-40 02 01 124 ПЗ

Студент: гр. 150501 Черноок А.Ю.

Руководитель: старший преподаватель
каф. ЭВМ Поденок Л.П.

Минск 2023

Учреждение образования
«Белорусский государственный университет информатики
и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ
Заведующий кафедрой
_____
(подпись)
_____2023г.

ЗАДАНИЕ
по курсовому проектированию

Студенту *Черноок Анастасие Юрьевне*
1. Тема проекта *Утилита контроля появления дубликатов в файловой системе с заменой их на жесткие ссылки и протоколирования фактов замены.*
2. Срок сдачи студентом законченного проекта *17 мая 2023 г.*
3. Исходные данные к проекту: *Язык программирования Си. Операционная система Linux. Задание - реализовать утилиту контроля появления дубликатов в файловой системе с заменой их на жесткие ссылки и протоколирования фактов замены*.
4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке):
1. Задание.
2. Введение.
3. Обзор методов и алгоритмов решения поставленной задачи.
4. Обоснование выбранных методов и алгоритмов.
5. Описание программы для программиста.
6. Описание алгоритмов решения задачи.
7. Руководство пользователя.
8. Заключение.
9. Литература.
10. Приложения.
5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)
1. Схема алгоритма main().
2. Схема алгоритма show_daemon_cond().
3. Схема алгоритма daemon_create().
4. Схема алгоритма file_digest_update().
5. Схема dir_run_save_files_in_db().

6. Консультант по проекту (с обозначением разделов проекта) *Поденок Л.П.*
7. Дата выдачи задания *15.02.2023г.*
8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):
1. Выбор задания. Разработка содержания пояснительной записки.
Перечень графического материала к 01.03 – 15 %;
разделы 2, 3 к 15.03 – 10 %;
разделы 4 к 15.04 – 20 %;
разделы 5 к 15.04 – 35 %;
раздел 6,7,8 к 15.05 – 5 %;
раздел 9 к 15.05 – 5%;
оформление пояснительной записки и графического материала к 17.05.23 – 10 %
Защита курсового проекта с 29.05.2023 по 09.06.2023г.
РУКОВОДИТЕЛЬ Поденок Л.П. _____
(подпись)

Задание принял к исполнению _____ Черноок А.Ю.
(дата и подпись студента)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время в мире существует огромное количество информации, которую нужно хранить и обрабатывать. Из-за растущего объема информации возникает все больше проблем, связанных с ее фильтрацией и контролем. От чего следует, что для пользователей компьютеров и других устройств становится все более актуальной задача мониторинга и управления данными.

Современная файловая система может содержать огромное количество файлов различных типов и форматов. К сожалению, это может привести к появлению их дубликатов, которые не только занимают дополнительное место на жестком диске, но и ухудшают производительность компьютера, затрудняют процесс управления файлами и их поиск.

Данная курсовая работа посвящена разработке утилиты для операционной системы `Linux`, которая должна осуществлять мониторинг файловой системы, контролировать появление дубликатов файлов в ней, заменять дубликаты на ссылки, а также протоколировать факты замены.

Использование жестких ссылок вместо копирования файлов должно позволить сохранить место на жестком диске и улучшить производительность компьютера. Кроме того, ссылки должны помочь пользователю более эффективно управлять своими данными и быстрее находить нужные файлы.

Для реализации вышеизложенной идеи в связи с мощными возможностями и высокой производительностью в качестве языка программирования принято решение использовать язык Си.

Си — это язык программирования, который используется для создания операционных систем, драйверов устройств и других программ, работающих на низком уровне.

Си является языком программирования, который позволяет создавать эффективный и оптимизированный код для системы. Он предоставляет разработчикам прямой доступ к памяти, что позволяет лучше контролировать процессы, происходящие в программе, а также ресурсы, которые та использует. Это делает его подходящим выбором для создания утилиты, работающей с файловой системой.

В целом, выбор языка Си для создания утилиты контроля дубликатов в файловой системе является оптимальным решением, так как он обеспечивает высокую производительность и простоту разработки, что позволит создать надежное и эффективное приложение.

1 ОБЗОР МЕТОДОВ И АЛГОРИТМОВ РЕШЕНИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ

1.1 Варианты реализации программы

Реализация утилиты для контроля дубликатов в файловой системе с заменой их на жесткие ссылки и протоколированием фактов замены может быть выполнена различными способами.

Данную программу возможно реализовать в виде консольного приложения, которое будет запускаться пользователем вручную при необходимости контроля файловой системы, а также иметь возможность управлять процессом обработки файлов. Пользователь сможет указать каталоги, которые необходимо обработать, и программа будет сканировать эти каталоги в поисках дубликатов файлов. При обнаружении дубликата, программа заменит его на ссылку и запишет соответствующую информацию в лог-файл. Однако необходимость постоянного вмешательства пользователя для запуска и повторного указания каталогов является существенным недостатком данного способа.

Следующий вариант решения задачи - реализация утилиты в виде программы-демона. Демон (daemon) – это программа, которая работает в фоновом режиме и выполняет определенные задачи. Реализация утилиты в виде демона позволяет контролировать файловую систему в режиме реального времени, обрабатывать новые файлы при их появлении и заменять дубликаты на ссылки без вмешательства пользователя лишний раз. Демон может периодически сканировать заданные каталоги на наличие дубликатов или обрабатывать происходящие в файловой системе события и автоматически выполнять замену файлов на ссылки. Этот вариант реализации способен устранить необходимость постоянного вмешательства пользователя и обеспечить более автоматизированный и удобный процесс контроля дубликатов.

1.2 Использование ссылок в качестве замены дубликатов

Сохранение дубликатов в файловой системе может привести к негативным последствиям. Они занимают дополнительное пространство на диске, из-за чего ресурсы системы используются неэффективно, особенно если размеры дублирующих файлов довольно большие. Также это усложняет процесс управления файлами, так как каждую копию нужно контролировать отдельно, что в свою очередь может привести к ошибкам и пропускам при обновлении или удалении файлов. Наличие дубликатов может привести к несогласованности данных, если разные копии одного и того же файла изменяются независимо друг от друга.

В данной ситуации уместно использование ссылок. Существует два типа ссылок: символические и жесткие.

Символические ссылки — это файлы, которые содержат ссылку на другой файл в файловой системе. При обращении к символической ссылке, система перенаправляет запрос к файлу, на который она ссылается. Их достоинством является возможность ссылаться на файлы, находящиеся в других каталогах, разделах диска или на других устройствах. Однако их важным недостатком является их уязвимость к удалению и перемещению, так как, если исходный файл, на который указывает символическая ссылка, будет удален или перемещен, ссылка станет недействительной, что может привести к проблемам, если другие программы или процессы полагаются на такую ссылку.

Жесткие ссылки — это другой тип ссылок, который создает ссылку на уже существующий файл в файловой системе. При создании жесткой ссылки, система создает новую запись в файловой системе, указывающую на те же данные, что и исходный файл. В этом случае, если исходный файл изменяется, изменения будут отображаться и в жесткой ссылке. Жесткие ссылки могут быть созданы только в пределах одной файловой системы, они не могут быть созданы на других разделах диска или устройствах.

1.3 Способы выявления дубликатов

Для отслеживания появления дубликатов файла можно использовать такие данные файла, как его имя и размер. Однако их недостаточно, так файлы самого разного содержания могут иметь одинаковый размер. А если файлы имеют одинаковое имя, поиск дубликатов теряет свой смысл.

В таком случае для наиболее точного выявления дубликата логично рассматривать содержимое файлов. Однако сравнение каждого бита файла может являться очень как время-, так и ресурсозатратным процессом. Для оптимизации проверки содержимого файла существует хэш. Хэш файла — это уникальный идентификатор файла, генерируемый на основе его содержимого.

Хэш вычисляется по специальному алгоритму. При необходимости вычисления хэша программист может как реализовать свой собственный, так и использовать готовые алгоритмы, которых на данный момент существует огромное количество.

Самыми распространенными алгоритмами хэширования можно назвать `MD5`, `SHA-1`, `SHA-2`. Каждый из них может быть более или менее подходящим для конкретного применения в зависимости от требований к безопасности и производительности программы.

`MD5` (`Message-Digest Algorithm 5`) — это криптографический алгоритм хэширования сообщений, который используется для проверки целостности данных и защиты от подделки. Преимущества `MD5` включают высокую скорость работы и малый размер хэш-значения, но он является устаревшим и наиболее уязвимым к коллизиям (явление, когда различные данные дают одинаковый хэш).

SHA (Secure Hash Algorithm) является семейством алгоритмов хэширования, которые преобразуют входные данные произвольной длины в фиксированный выходной хэш.

SHA-1 такой же быстрый, как и MD5, и имеет более безопасную хэш-функцию, но на данный момент также стал довольно сильно уязвимым к коллизиям и устарелым.

SHA-2 (включая SHA-256 и SHA-512) считается более безопасным, чем MD5 и SHA-1, так как он имеет большую длину хэш-значения и более сложную хэш-функцию. Его главным недостатком является более медленная скорость работы в сравнении с MD5 и SHA-1.

1.4 Хранение данных

В ходе контроля появления дубликатов файлов нужно хранить ранее обработанную информацию.

Первым вариантом является хранение необходимой информации в текстовом или бинарном файле по своей разработанной структуре. При правильном проектировании данный способ может максимально сэкономить время и ресурсы, затрачиваемые программой. Существенным недостатком является время программиста, затраченное на должное проектирование и реализацию.

Оптимальным и рациональным путем хранения данных является вариант использование баз данных. Они являются эффективным инструментом для хранения, организации и быстрого доступа к большим объемам информации. Среди баз данных, с которыми можно работать на языке си, можно выделить нижеприведенные.

SQLite — это компактная база данных, которая хранит данные в локальном файле и не требует сервер. Она предназначена для встраивания в как в мобильные, так и десктопные приложения. Преимущества SQLite включают в себя простоту использования, легковесность и низкие требования к ресурсам системы. Недостатками являются низкая производительность и ограниченность масштабирования.

MySQL — это более тяжеловесная реляционная база данных, которая требует установки сервера на хост-системе. Она предоставляет множество возможностей для работы с базами данных, включая SQL-запросы, транзакции, события и т.д. Преимущества MySQL включают в себя высокую производительность, масштабируемость и поддержку больших объемов данных. Недостатки - требовательность к ресурсам системы и сложность настройки.

Berkeley DB — это встраиваемая нереляционная база данных, которая обеспечивает быстрое и надежное хранение данных в приложении. Преимущества Berkeley DB включают в себя высокую производительность и легковесность. Недостатки - ограниченность функциональности, отсутствие поддержки SQL-запросов и ограниченная масштабируемость при работе с большими объемами данных.

1.5 Проверка файловой системы

Контролировать появление дубликатов в файловой системе можно двумя способами.

Первый заключается в том, чтобы повторно через определенные промежутки времени сканировать отслеживаемые каталоги. Плюсом является возможность настроить систему контроля обнаружения дубликатов таким образом, чтобы она выполнялась в определенное время или через заданные промежутки времени, когда в системе “накопятся” файлы, что могут оказаться дубликатами существующих. Однако, следует учитывать, что регулярное сканирование может быть достаточно ресурсоемким процессом, особенно если каталоги содержат большое количество файлов. Это может замедлить работу компьютера.

Второй способ контроля появления дубликатов заключается в отслеживании событий, происходящих в файловой системе. В отличие от первого подхода, при котором система производит сканирование каталогов с определенной периодичностью, второй подход основывается на мониторинге событий, происходящих в файловой системе, и немедленной реакции на них. Преимуществом этого подхода является быстрота обнаружения дубликатов файлов в реальном времени, что позволяет предотвратить накопление лишних файлов даже на небольшой промежуток времени. Проблемой и недостатком могут стать инструменты, необходимые для реализации данного метода. Для реализации этого подхода могут использоваться как встроенные средства операционной системы, так и специализированные программы.

2 ОБОСНОВАНИЕ ВЫБРАННЫХ МЕТОДОВ И АЛГОРИТМОВ

2.1 Реализация программы

Выбор демона для реализации решения задачи является рациональным и подходящим, поскольку демон обладает рядом преимуществ, которые делают его более эффективным и гибким инструментом.

Во-первых, возможность управления работой демона через специальные команды из консоли значительно облегчает процесс мониторинга и управления утилитой. Это позволяет быстро и эффективно изменять настройки и запускать/останавливать процесс обработки файловой системы.

Во-вторых, демон может быть реализован таким образом, чтобы он не занимал много ресурсов системы и не влиял на ее производительность. Это особенно важно для больших файловых систем с большим количеством файлов и папок, которые могут сильно нагружать процессор и оперативную память компьютера.

В-третьих, демон обеспечивает работу приложения в фоновом режиме, что позволяет контролировать файловую систему в режиме реального времени.

Выбор демона является наиболее рациональным и подходящим для решения задачи.

2.2 Подходящий тип ссылок

При выборе между жесткими и символическими ссылками предпочтителен вариант использования первых. Так как если переместить или переименовать файл, на который указывает символическая ссылка, сама ссылка станет недействительной, и при попытке использовать ее произойдет ошибка.

Жесткие ссылки обладают более высокой степенью надежности по сравнению с символическими. Если исходный файл перемещается или переименовывается, то жесткая ссылка остается действительной и продолжает указывать на исходный файл.

Таким образом, использование жестких ссылок может существенно сэкономить место на диске и обеспечить целостность данных в системе.

2.3 Проверка на дубликат

Наиболее подходящим способом отслеживания дубликатов является проверка размера или хэша файла. Размер файла — это основные данные файла, которые можно быстро и легко сравнить, чтобы определить, являются ли два файла дубликатами.

Сравнение по хэшу является наиболее точным способом идентификации дубликатов файлов, потому что он учитывает содержимое файла, что делает

невозможным случайную генерацию одинаковых хэшей для двух разных файлов.

Однако, вычисление хэша для большого количества файлов может занять много времени и ресурсов. Поэтому для оптимальной производительности, желательно совмещать оба параметра для поиска дубликатов файлов.

Для вычисления хэша файлов в данном проекте рационально принятие решения об использования такого алгоритма хэширования, как SHA.

Расчет хэша по данному алгоритму может быть произведён с помощью средств библиотеки OpenSSL.

Библиотека OpenSSL — это бесплатный набор криптографических инструментов и библиотек с открытым исходным кодом, который предоставляет разработчикам инструменты для реализации безопасности и шифрования в их приложениях.

2.4 База данных

Самым оптимальным путем хранения данных является использование баз данных. В контексте данного курсового проекта, их использование может быть полезно для хранения информации как о каталогах, так и об обработанных файлах.

Наиболее подходящей базой данных является Berkeley DB. Для её использования не нужно устанавливать сервер, что абсолютно не необходимо в рамках текущего проекта. Преимуществами использования данной базы данных является высокая скорость работы, возможность многопоточной обработки данных, а также надежность и целостность хранения информации.

Для работы с базой данных Berkeley DB в языке Си необходимо использовать библиотеку “libdb”, которая предоставляет интерфейс для доступа к базе данных.

Библиотека “libdb” хранит данные как ключ-значение и поддерживает несколько различных типов баз данных, включая B-tree, который прекрасно подходит для структурированного хранения информации об их именах и размерах файлов, что делает Berkeley DB отличным кандидатом для использования в данном курсовом проекте с целью хранения необходимой для функционирования разрабатываемой утилиты информации.

2.5 Обработка событий

При выборе между периодичным сканированием нужных каталогов и обработкой событий в реальном времени более предпочтительным является второй вариант за счет мгновенного реагирования и меньших затрат ресурсов.

Так как реализовываемая программа разрабатывается под операционную систему Linux, то подходящим инструментом для данной задачи является такое средство, как Inotify.

Inotify — это механизм ядра Linux, который позволяет приложениям получать “уведомления” об изменениях в файловой системе. С помощью Inotify можно отслеживать такие события, как создание, изменение или удаление файлов или каталогов, а также получать уведомления об их перемещении или переименовании.

Inotify позволяет создать специальный файловый дескриптор, который связывается с файлом или каталогом, за изменениями которого нужно следить. Когда происходит событие, связанное с этим файлом или каталогом, Inotify генерирует уведомление и отправляет его на ранее созданный файловый дескриптор.

Inotify в разрабатываемой утилите прекрасно подходит для использования демоном, что должен контролировать появление дубликатов файлов.

3 ЦЕЛИ И ПОСТАНОВКА ЗАДАЧИ

Таким образом исходя из вышесказанного следует выделить, что целью данного курсового проекта является создание на языке Си для работы с операционной системой Linux приложения-демона, работающего в автономном режиме и следящего за появлением дубликатов в выбранных пользователем каталогах. Дубликаты выявляются за счёт сравнения размеров и хэшей файлов.

Рациональным решением является создание также приложения для управления демоном: программы-контроллера, с помощью которого пользователь сможет добавлять/удалять каталоги для мониторинга и “включать”/“выключать” демона. Так как основной задачей курсового проекта является создание утилиты контроля появления дубликатов, а задачи контроллера вторичны, минималистичного консольного приложения достаточно.

При нахождении дубликатов демон должен заменять те на жесткие ссылки, а также протоколировать факт замены. Поиск дубликатов должен осуществляться за счет сравнения размеров и хэшей файлов, что позволит обнаруживать дубликаты с разными именами. Для сохранения указанных данных будет применяться база данных Berkeley DB.

4 ОПИСАНИЕ ПРОГРАММЫ ДЛЯ ПРОГРАММИСТА

4.1 Структура программы

Разрабатываемая программа состоит из двух основных частей: “Контроллера” и “Демона”.

“Контроллер” – та часть программы, что взаимодействует с пользователем и позволяет тому выполнять следующие опции:

1) Добавить каталог для мониторинга. Добавление осуществляется за счёт ввода имени интересующего каталога с последующей проверкой на наличие того в системе.

2) Вывод всех каталогов, находящихся под наблюдением.

3) Удалить каталог из списка отслеживаемых. Вначале производится вывод всех каталогов под наблюдением, далее пользователю предоставляется выбор на удаление.

4) “Включить”/“выключить” утилиту обнаружения появления дубликатов.

5) Просмотреть, активен демон или нет.

Также пара скрытых из меню пользователя, но доступных функций:

1) Вывод всех отслеживаемых файлов на появление дубликата.

2) Вывод всех отслеживаемых поддиректорий.

“Демон” – та часть программы, что отслеживает изменения в файловой системе и контролирует появление дубликатов. Среди её задач можно выделить следующие:

1) Проход по выбранным пользователем каталогам и заполнение базы данных, хранящей информацию об именах и размерах файлов, с попутной заменой обнаруженных дубликатов на жесткие ссылки.

2) Отслеживание следующих событий и принятие соответствующих мер:
- появление файла с дальнейшей проверкой на дубликат;
- изменение содержимого файла с проверкой на дубликат;
- переименовывание файла с обновлением его имени в базе;
- удаление файла с удалением его данных из базы данных;
- появление каталога с дальнейшей проверкой на наличие дубликатов;
- переименовывание каталога с обновлением имен его файлов в базе;
- удаление каталога с удалением данных его файлов из базы данных.

4.2 Структура проекта

Курсовой проект представляет собой каталог, состоящий из нескольких подкаталогов, каждый из которых хранит определенные данные для функционирования утилиты:

1) В каталоге `src` находится исходный код программы. Он включает в себя файлы, в которых реализован функционал для работы с кэшем через библиотеку `OpenSSL`, функционал для работы с базами данных `libdb`, а также

функционал Inotify для мониторинга изменений в файловой системе. Также в данном каталоге находится makefile, необходимый для сборки проекта.

2) В каталоге build хранится скомпилированный файл программы, с помощью которого пользователь работает с приложением. Также в данном каталоге после создания программой находится файл logfile.txt, что протоколирует работу утилиты контроля появления дубликатов в файловой системе с заменой тех на жесткие ссылки.

3) Каталог database содержит файлы баз данных, которые используются утилитой для хранения информации о файлах и отслеживаемых каталогах.

4) В каталоге docs находится файл с информацией о проекте, принципе его работы, структуре, а также руководством пользователя.

5) Каталог lib содержит необходимые для сборки проекта библиотеки, не предустановленные в системе Linux. Если программист желает пересобрать проект, а не использовать готовый скомпилированный файл приложения, он должен самостоятельно добавить эти библиотеки.

6) Каталог test содержит файлы и каталоги, используемые для проверки правильности работы утилиты. Каталог разбит на несколько подкаталогов, в каждом из которых находятся тесты, относящиеся к определенной функциональности утилиты.

4.3 Структура файлов

Кроме файлов, что содержат исходный код проекта, и тех, что используются в тестах работы утилиты, проект содержит следующие файлы:

1) directories_database.db. Данный файл является базой данных, содержащей имена каталогов, выбранных пользователем для мониторинга появления дубликатов.

2) files_database.db. Данный файл является базой данных, содержащей информацию о файлах отслеживаемых каталогов, появление дубликатов которых находится под контролем. В качестве ключа используется размер файла, а в качестве значения – его имя.

3) inot_subdirectories_database.db. Данный файл является базой данных, содержащей информацию о подкаталогах отслеживаемых каталогов, необходимого для системы Inotify, что не предусматривает рекурсивного обхода выбранных каталогов. В качестве ключа используется переменная, в которую сохраняется идентификатор (watch descriptor) для отслеживаемого каталога, а в качестве значения – его имя.

4) daemon_stat.pid. Данный файл содержит pid запущенного демона. Данный файл необходим, так как pid нужен для связи с демоном, а поскольку pid демона знает только контроллер, создавший его, то данное значение важно сохранить на постоянном носителе.

5) inot_fd. Данный файл содержит значение инстанса Inotify.

6) `logfile.txt`. Данный файл содержит информацию о времени и факте замены дубликата на жесткую ссылку, а также информацию о проблеме работы утилиты, если та возникла.

4.4 Тест программы

Так как в данном проекте используются не предустановленные системой Linux библиотеки, для его сборки необходимо из `lib` добавить каталоги `libdb` и `openssl` в каталог `include`, хранящий системные заголовочные файлы. Проект разрабатывается на системе Linux Fedora. В данной операционной системе нужным является каталог `/usr/include`. Он может разниться в зависимости от используемой операционной системы.

После запуска программы `./dsnr_controller` в консоли выводится меню:

```
~~Duplicates Searcher & Replacer~~
c - DSNR Daemon condition
t - Turn On/Off DSNR Daemon

l - List of Monitored Directories
a - Add Directory for monitoring
d - Delete Directory for monitoring

q – Quit
```

При попытке запуска выводится предупреждение об отсутствии каталогов для отслеживания.

```
Your Choice:
t
NO DIRECTORIES TO MONITOR
```

Далее представлен процесс ввода каталогов для мониторинга:

```
Your Choice:
a
Enter Directory Name and press Enter:
../test/dups/

Directory:
[/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/]  successfully added for monitoring

Your Choice:
a
Enter Directory Name and press Enter:
../test
Entered Directory has a subdirectory among of already monitored ones.

Your Choice:
```

```
    a
    Enter Directory Name and press Enter:
    ../test/dupser

    Directory:
    [/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupser/]
successfully added for monitoring

    Your Choice:
    a
    Enter Directory Name and press Enter:
    ../test/dupsesss
    Entered file is not found.

    Your Choice:
    a
    Enter Directory Name and press Enter:
    ../test/dupses

    Directory:
    [/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupses/]
successfully added for monitoring
```

Далее происходит вывод списка наблюдаемых каталогов, запуск демона и выход из контроллера.

```
    Your Choice:
    l

    List of Monitored Directories:
    [../database/directories_database.db] contains 3 records
    1. /home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/
    2. /home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupser/
    3. /home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupses/

    Your Choice:
    t
    Daemon is switched on

    Your Choice:
    q
```

После старта демона тот обрабатывает файлы в заданных каталогах и при наличии дубликатов заменяет те на жесткие ссылки, протоколируя факты замены в файле `logfile.txt`:

```
    Fri May 12 23:45:33 2023:
    Hard link on
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/aaa.txt
    by path
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/ddd/aa1.txt is
created

    Fri May 12 23:45:33 2023:
```

```
    Hard link on
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/evenn122
    by path /home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/evenrrcp
is created

    Fri May 12 23:45:33 2023:
    Hard link on
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/plll3/nnnn
    by path
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupses/ren0.txt is
created
```

После повторного запуска контроллера при запросе вывести отслеживаемые файлы и подкаталоги, контроллер выведет следующее:

```
    Your Choice:
    f
    [../database/files_database.db] contains 10 records
    1. 14 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/evenn122
    2. 16 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupser/kartew/filrrr
    3. 19 -- /home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupser/x34
    4. 21 -- /home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupser/aswqa
    5. 25 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/plll3/nnnn
    6. 25792 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/dirwalk
    7. 39 -- /home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/aaa.txt
    8. 39 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/ddd/111.txt
    9. 49 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupses/c5533.txt
    10. 54 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/ddd/aa22.txt

    Your Choice
    s
    [../database/inot_subdirectories_database.db] contains 7 records
    1. 1 -- /home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/
    2. 2 -- /home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/ddd/
    3. 3 -- /home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/ddd/555/
    4. 4 -- /home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/plll3/
    5. 5 -- /home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupser/
    6. 6 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupser/kartew/
    7. 7 -- /home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupses/
```

После переименовывания файла `aswqa` на `nassa`, строка с тем в спискефайлов заменяется на:

```
    4. 21 -- /home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupser/nassa
```

После копирования каталога tocop2 в plll3 списки файлов и подкаталогов обновляются до следующих:

```
    Your Choice:
    f
    [../database/files_database.db] contains 12 records
    1. 13 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/plll3/tocop2/rrr/55
    2. 13 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/plll3/tocop2/44
    3. 14 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/evenn122
    4. 16 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupser/kartew/filrrr
    5. 19 -- /home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupser/x34
    6. 21 -- /home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupser/nassa
    7. 25 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/plll3/nnnn
    8. 25792 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/dirwalk
    9. 39 -- /home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/aaa.txt
    10. 39 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/ddd/111.txt
    11. 49 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupses/c5533.txt
    12. 54 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/ddd/aa22.txt

    Your Choice:
    s
    [../database/inot_subdirectories_database.db] contains 9 records
    1. 1 -- /home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/
    2. 2 -- /home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/ddd/
    3. 3 -- /home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/ddd/555/
    4. 4 -- /home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/plll3/
    5. 5 -- /home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupser/
    6. 6 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupser/kartew/
    7. 7 -- /home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupses/
    8. 8 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/plll3/tocop2/
    9. 9 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/plll3/tocop2/rrr/
```

Если tocop2 переименовать в cope223, а из каталога plll3 удалить файл nnnn, информация обновится до следующей:

```
    Your Choice:
    f
    [../database/files_database.db] contains 11 records
    1. 13 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/plll3/cope234/44
    2. 13 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/plll3/cope234/rrr/55
```

```
    3. 14 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/evenn122
    4. 16 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupser/kartew/filrrr
    5. 19 -- /home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupser/x34
    6. 21 -- /home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupser/nassa
    7. 25792 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/dirwalk
    8. 39 -- /home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/aaa.txt
    9. 39 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/ddd/111.txt
    10. 49 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupses/c5533.txt
    11. 54 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/ddd/aa22.txt
```

Далее в каталог `dupses` вставляем каталог `tocop`, содержащий файл дубликат. Информация обновляется до следующей:

```
    Your Choice:
    f
    [../database/files_database.db] contains 13 records
    1. 10 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupses/tocop/rrr/55
    2. 12 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupses/tocop/765
    3. 13 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/plll3/cope234/44
    4. 13 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/plll3/cope234/rrr/55
    5. 14 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/evenn122
    6. 16 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupser/kartew/filrrr
    7. 19 -- /home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupser/x34
    8. 21 -- /home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupser/nassa
    9. 25792 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/dirwalk
    10. 39 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/aaa.txt
    11. 39 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/ddd/111.txt
    12. 49 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupses/c5533.txt
    13. 54 --
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dups/ddd/aa22.txt
```

А в файл `logfile.txt` добавляется информация о замене дубликата на ссылку:

```
    Fri May 12 23:56:33 2023:
    Hard link on
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupses/tocop/rrr/55
```

```
    by path
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupses/tocop/44 is
created
```

Далее в каталоге `dupses` создаем новый файл `exampo`. В списке файлов появляется новая запись о нем:

```
    0 -- /home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupses/exampo
```

После его редактирования появляется новая запись:

```
    13 -- /home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupses/exampo
```

Если отредактировать ранее созданный файл так, чтобы он стал копией другого, он станет жесткой ссылкой, а факт замены запротоколируется в файл `logfile.txt`:

```
    Fri May 12 23:58:24 2023:
    Hard link on
/home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupser/nassa
    by path /home/niks0/MNFolder/oslabs/kurs_w_dirs/test/dupses/exampo
is created
```

В конце представлен процесс проверки состояния демона и его выключения:

```
    Your Choice:
    c
    Daemon is on
    It's pid:7270

    Your Choice:
    t
    Daemon is switched off

    Your Choice:
    q
    [niks0@fedora build]$
```

5 ОПИСАНИЕ АЛГОРИТМОВ РЕШЕНИЯ ЗАДАЧИ

5.1 Описание схем алгоритмов

Ниже представлено описание схем алгоритмов одних из самых важных функций в проекте.

Схема алгоритма функции `main()` – основной функции проекта, представлена в приложении А.

Схема алгоритма функции `show_daemon_cond()`, предназначенной для вывода состояния демона, представлена в приложении Б.

Схема алгоритма функции `daemon_create()`, отвечающей за создание демона, представлена в приложении В.

Схема алгоритма функции `file_digest_update()`, предназначенной для заполнения контекста данными файла для дальнейшего вычисления хэша, представлена в приложении Г.

Схема алгоритма функции `dir_run_save_files_in_db()`, предназначенной для рекурсивного прохода по каталогу и обработки его файлов, представлена в приложении Д.

5.2 Описание алгоритмов

Ниже представлены алгоритмы одних из самых значимых функций в проекте.

Алгоритм функции `daemon_start()`, предназначенной для запуска демона:

1) Начало.

2) Очистить базу данных `files_database.db` с помощью функции `clear_database()`.

3) Очистить базу данных `inot_subdirectories_database.db` с помощью функции `clear_database()`.

4) Обработать файлы заданных пользователем каталогов с помощью функции `chosen_dirs_process()`. Если функция вернула не 0, перейти к пункту 6.

5) Вывести сообщение об отсутствии каталогов в базе данных `directories_database.db`. Перейти к пункту 8.

6) Запустить демона с помощью функции `daemon_create()`.

7) Вывести сообщение об успешном старте демона.

8) Конец.

Алгоритм функции `inot_events_proccessing()`, отвечающей за обработку событий, происходящих во время работы демона:

1) Начало.

2) Промежуточные данные:

`Char buffer [BUF_LEN]` – буфер, в который записывается группа событий;

Int i – индекс для прохода по данному буферу;

Int length – “длина” произошедшей группы событий;

Int fd – переменная, хранящая инстанс Inotify;

Char dir_str [FILE_NAME_LEN] – символьный массив для хранения имени каталога, событие в котором произошло.

Char wd_str [VALUE_STR_LEN] – символьный массив для хранения значения идентификатора (watch descriptor) каталога, в котором произошло событие;

Char file_str [FILE_NAME_LEN] – символьный массив для хранения имени файла, событие связанное с которым произошло;

Struct inotify_event * event – указатель на структуру, содержащую информацию о произошедшем событии.

3) Прочитать все события, произошедшие во время подготовки к старту демона (создание жестких ссылок при их наличии).

4) Начало цикла обработки событий.

5) i = 0.

6) Прочитать произошедшую группу событий и присвоить её длину length с помощью функции read(fd, buffer, BUF_LEN).

7) Начало цикла обработки произошедшей группы событий.

8) Прочитать текущее событие.

9) Перевести значение идентификатора (watch descriptor) в символьный массив wd_str и с помощью функции get_data_by_key_in_db_tree() получить и записать в dir_str имя каталога, событие в котором произошло.

10) Используя dir_str и имя файла, событие, связанное с которым произошло, записать в file_str полное имя данного файла.

11) Если событие произошло не в каталоге, перейти к пункту 15.

12) Если событие IN_CREATE или IN_MOVED_TO, обработать каталог с помощью функции dir_run_save_files_in_db().

13) Если событие IN_DELETE, обработать его с помощью функции del_dir_process().

14) Перейти к пункту 17.

15) Если событие IN_CLOSE_WRITE или IN_MOVED_TO, обработать файл с помощью функции in_write_closed_process().

16) Если событие IN_MOVED_FROM или IN_DELETE, обработать файл с помощью функции del_process().

17) Увеличить значение i на “длину” обрабатываемого события.

18) Если i < length, перейти к пункту 7.

19) Перейти к пункту 4.

20) Конец.

Бесконечный цикл прерывается по приходу сигнала, требующего завершить работу демона.

Алгоритм функции add_directory_for_monit(), предназначенной для добавления каталога в список отслеживаемых:

1) Начало.

2) Промежуточные данные:

Char dir_name – символьный массив, где будет сохранено имя введенного каталога;

Struct stat sb – структура с информацией о файле.

3) Произвести ввод названия каталога с помощью функции str_input( ). Если та вернула 0, перейти к пункту 9.

4) Проверить наличие файла и сохранить информацию о нем в sb с помощью функции lstat( ). Если та вернула -1, перейти к пункту 9.

5) Проверить тип файла, проверив поле st_mode sb. Если тот не является каталогом перейти к пункту 9.

6) Получить полное имя каталога с помощью функции get_full_dir_path( ).

7) Проверить введенный каталог на наличие в базе данных с помощью функции check_key_presence_in_db_tree( ). Если та вернула не 0, перейти к пункту 9.

8) Проверить каталог с помощью функции check_dirs_on_sub( ). Если та вернула 1 или 2, перейти к пункту 9 (1 говорит о том, что введенный каталог является подкаталогом уже присутствующего в базе, 2 – о том, что тот имеет подкаталоги в базе).

9) Конец.

Алгоритм функции files_hash_compare( ), предназначенной для сравнения хэшей двух файлов:

1) Начало.

2) Входные данные:

Char * file_name1 – указатель на символьный массив, содержащий имя первого файла для сравнения;

Char * file_name2 – указатель на символьный массив, содержащий имя второго файла для сравнения.

Промежуточные данные:

EVP_MD_CTX * mdctx1 – указатель на контекст данными первого файла для дальнейшего вычисления хэша;

EVP_MD_CTX * mdctx2 - указатель на контекст данными второго файла для дальнейшего вычисления хэша;

Const EVP_MD * md – указатель на структуру, определяющую метод хэширования.

Выходные данные:

Unsigned char md1_value[EVP_MAX_MD_SIZE] – символьный массив, содержащий высчитанный хэш первого файла;

Unsigned char md2_value[EVP_MAX_MD_SIZE] – символьный массив, содержащий высчитанный хэш второго файла;

Unsigned int md1_len – длина хэша первого файла;

Unsigned int md2_len - длина хэша второго файла;

Int are_equal – переменная для регистрации равенства хэшей.

3) Определить метод хэширования SHA256 с помощью функции EVP_get_digestbyname( ).

4) Инициализация контекста mdctx1 для первого файла с помощью функции EVP_MD_CTX_new( ).

5) Инициализация контекста mdctx2 для второго файла с помощью функции EVP_MD_CTX_new( ).

6) Заполнить контекст mdctx1, используя содержимое файла file_name1 с помощью функции file_digest_update( ).

7) Заполнить контекст mdctx2, используя содержимое файла file_name2 с помощью функции file_digest_update( ).

8) Сформировать хэш первого файла и занести его в md1_value с помощью функции EVP_DigestFinal_ex( ), также получив длину этого хэша md1_len.

9) Сформировать хэш второго файла и занести его в md2_value с помощью функции EVP_DigestFinal_ex( ), также получив длину этого хэша md2_len.

10) Очистить данные контекста mdctx1 с помощью функции EVP_MD_CTX_free( ).

11) Очистить данные контекста mdctx2 с помощью функции EVP_MD_CTX_free( ).

12) Проверить равенство хэшей, используя функцию memcmp( ). Результат занести в are_equal.

13) Вернуть are_equal.

14) Конец.

Алгоритм функции file_processing( ), предназначенной для обработки файла:

1) Начало.

2) Входные данные:

Struct stat * s - структура с информацией об обрабатываемом файле;

Char * full_f_name – имя обрабатываемого файла;

Промежуточные данные:

Char size_str [VALUE_STR_SIZE] - символьный массив для хранения размера обрабатываемого файла;

DB * dbp – указатель на базу данных, хранящую информацию об обработанных файлах;

DBT kkey – структура для хранения информации об очередном ключе из базы данных dbp;

DBT ddata – структура для хранения информации об очередном значении ключа kkey;

DBC * cursor – указатель на курсор (итератор) для прохождения по базе данных dbp;

Unsigned same – переменная для регистрации равенства рассматриваемых файлов.

3) Проверить тип файла, проверив поле st_mode s. Если тот не является обычным файлом, перейти к пункту 17.

4) Получить из поля st_size s размер рассматриваемого файла и перевести его в символьный массив size_str с помощью функции sprintf( ).

5) Проверить наличие файла с таким же размером в базе данных с файлами с помощью функции check_key_presence_in_db_tree( ). Если та вернула 0, перейти к пункту 18, так как файл с таким же размером отсутствует в базе.

6) Открыть базу данных dbp с помощью функции open_tree_database( ).

7) Инициализировать необходимые значения ключа kkey данными size_str.

8) Инициализировать курсор cursor и установить его на позицию первого элемента, соответствующего ключу kkey с помощью метода структуры DBC get( ) и флага DB_SET.

9) Присвоение same значения 0.

10) Сравнить хэши файлов full_f_name и (char *) ddata.data с помощью функции files_hash_compare( ). Если та вернула не 0, перейти к пункту 14.

11) Присвоить same значение 1.

12) Заменить рассматриваемый файл на жесткую ссылку с помощью функции link_create( ).

13) Перейти к пункту 16.

14) Перейти к следующему элементу в базе с помощью метода структуры DBC get( ) и флага DB_NEXT.

15) Сравнить символьные массивы с размером рассматриваемых файлов. Если те равны, перейти к пункту 10.

16) Проверить значение same. Если то равно 0, перейти к пункту 18.

17) Вернуть 1, что говорит об отсутствии необходимости добавления информации о рассматриваемом файле в базу данных с файлами. Перейти к пункту 19.

18) Вернуть 0, что говорит о необходимости добавления информации о рассматриваемом файле в базу данных с файлами.

19) Конец.

5.3 Код программы

Код программы представлен в приложении Е.

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Распаковать архив и сохранить каталог `dsnr_proj`.

Для начала работы утилиты из каталога `build` запустить файл с именем `./dsnr_controller` (при необходимости изменить права доступа).

Для выполнения действия из списка ввести соответствующий символ, а затем `Enter`.

Для добавления каталога ввести опцию ‘`a`’, затем либо полное, либо относительное имя каталога.

Для удаления каталога из списка отслеживаемых ввести опцию ‘`d`’, затем номер интересующего каталога из списка.

Опция ‘`t`’ запускает процесс контроля появления дубликатов, если тот был не активен, в противном случае завершает его.

Проверить процесс контроля появления дубликатов возможно при помощи опции ‘`c`’.

С помощью опции ‘`q`’ происходит выход из контроллера без влияния на процесс контроля появления дубликатов.

Проверить результаты работы утилиты можно в файле `logfile.txt` каталога `build`.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсового проекта были закреплены и углублены теоретические знания в области программирования на языке Си. Были получены знания и навыки в применении таких технологий, как хэширование данных при помощи алгоритма SHA-256 с использованием библиотеки OpenSSL, отслеживание событий в файловой системе операционной системы Linux с помощью механизма Inotify, а также обработка информации за счёт использования базы данных BerkeleyDB.

Была решена поставленная задача, а именно: была разработана утилита контроля появления дубликатов в файловой системе с заменой их на жесткие ссылки и протоколирования фактов замены. Основная её цель заключается в оптимизации использования места на жестких дисках и предотвращении дублирования информации в файловой системе.

Программа дает возможность создавать демона, что вначале проверяет указанные пользователем каталоги на наличие одинаковых по содержанию файлов, а далее следит за такими событиями, как создание, редактирование, перемещение, переименовывание и удаление файлов с последующей обработкой.

Разработанная утилита имеет потенциал для дальнейшего развития. Файловая система является очень сложной, и невозможно учесть все её нюансы. Существует огромный пласт из возможностей как для усовершенствования работы программы, так и для расширения её функционала.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Конструирование программ и языки программирования : метод. указания по курсовому проектированию для студентов специальности 1-40 02 01 «Вычисл. машины, системы и сети» всех форм обучения / сост. А. В. Бушкевич [и др.]. – Минск : БГУИР, 2010. – 30 с. : ил.

[2] Таненбаум, Э. Современные операционные системы / Э. Таненбаум, Х. Бос. – 4-е изд. – Санкт-Петербург : Питер, 2021. – 1120 с. : ил.

[3] Стивенс, У. Р. UNIX. Профессиональное программирование / У. Р. Стивенс, С. А. Раго ; пер. А. Киселева. – 2-е изд. – Санкт-Петербург : Символ-плюс, 2007. – 1040 с. : ил.

[4] Рочкинд, М. Д. Программирование для UNIX / М. Д. Рочкинд. – 2-е изд., перераб. и доп. – Санкт-Петербург : Русская Редакция : БХВ-Петербург, 2005. – 704 с.

[5] The Geek Stuff [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.thegeekstuff.com/2012/02/c-daemon-process.

[6] MD5 Hash Generator [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.md5hashgenerator.com.

[7] TProger [Электронный ресурс]. – Электронные данные. – Режим доступа: https://tproger.ru/translations/sha-2-step-by-step.

[8] WikiOpenSSL [Электронный ресурс]. – Электронные данные. – Режим доступа: https://wiki.openssl.org/index.php/Command_Line_Utilities.

[9] OpenSSL [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.openssl.org/docs/man3.1/man3/EVP_Q_digest.html.

[10] Man7 [Электронный ресурс]. – Электронные данные. – Режим доступа: https://man7.org/linux/man-pages/man7/inotify.7.html.

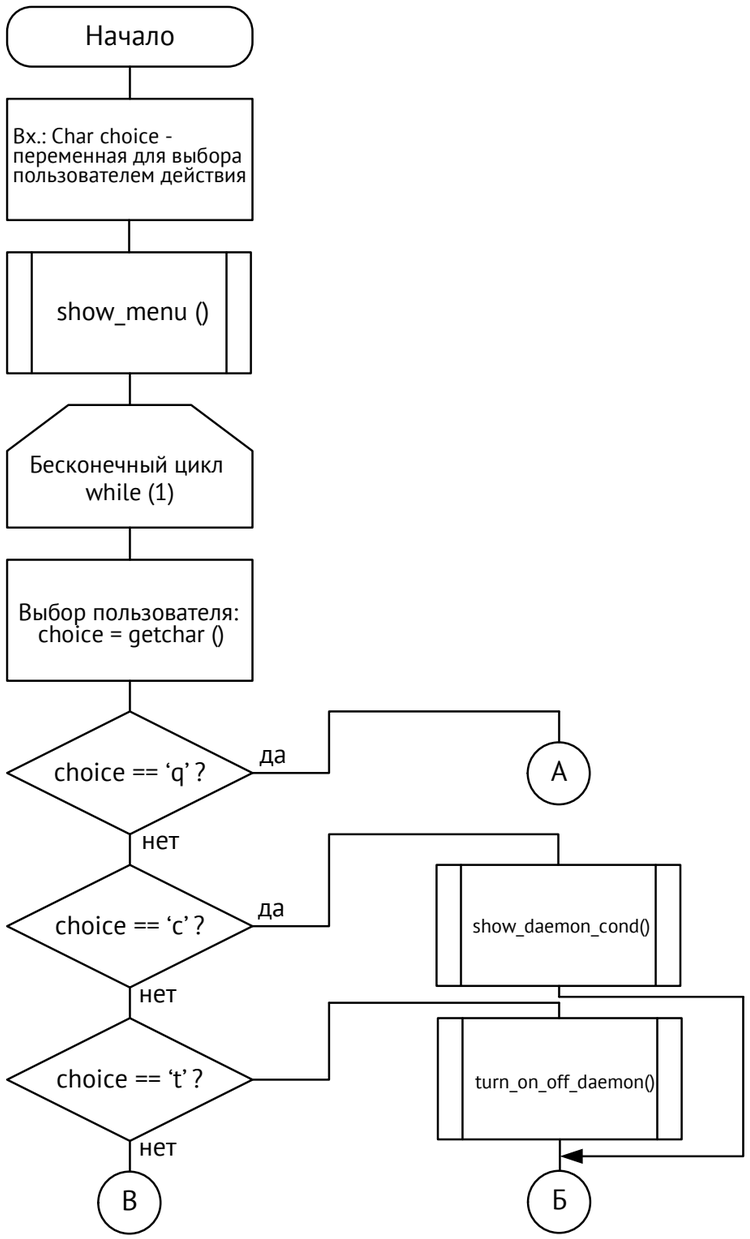
[11] MySQL [Электронный ресурс]. – Электронные данные. – Режим доступа: https://dev.mysql.com/doc/c-api/8.0/en.

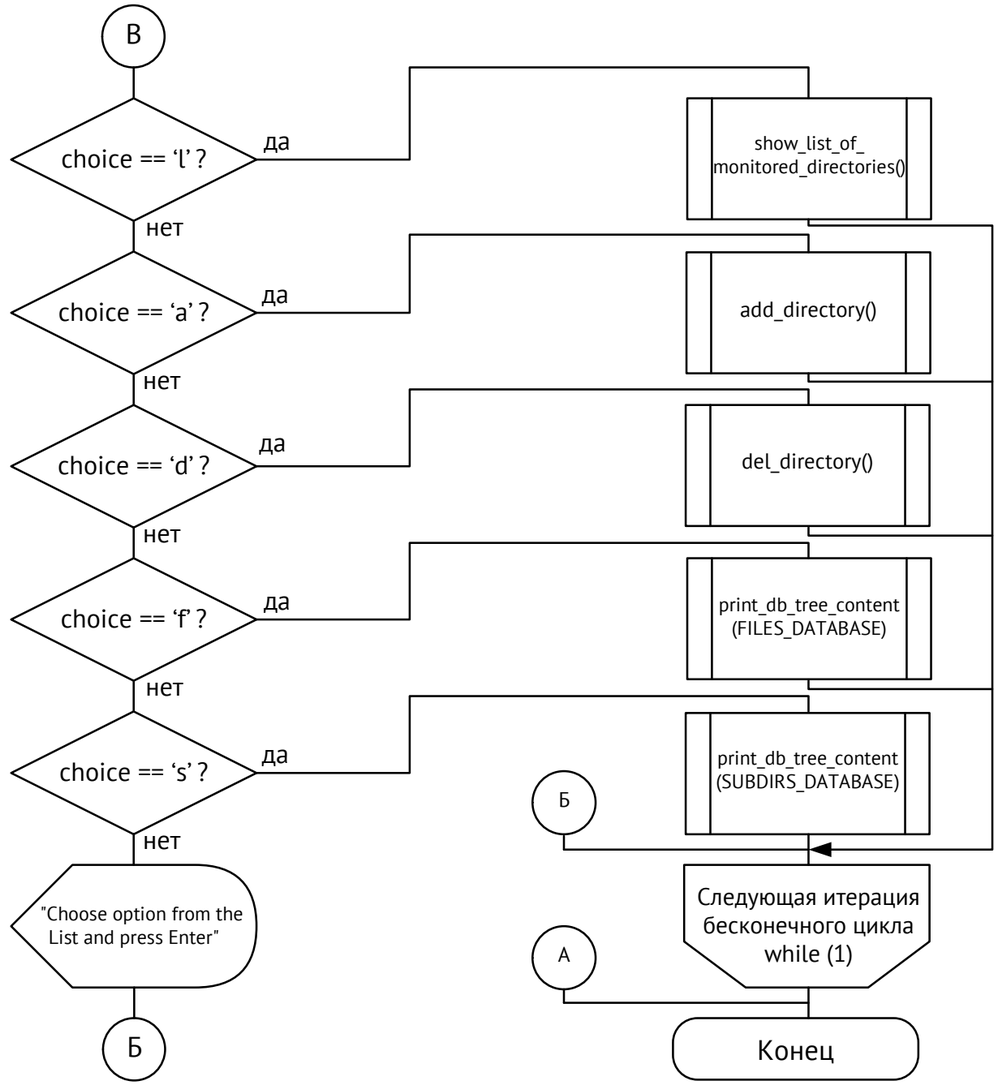
[12] SQLite [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.sqlite.org/cintro.html.

[13] Oracle [Электронный ресурс]. – Электронные данные. – Режим доступа: https://docs.oracle.com/cd/E17276_01/html/programmer_reference.html.

ПРИЛОЖЕНИЕ А
(обязательное)

Схема алгоритма функции `main()`





					ГУИР.400201.124 ПД1			
					Схема алгоритма функции main ()	Лит.	Масса	Масштаб
						У		
Изм	Лист	№ документа	Подпись	Дата		Лист	Листов 1	
Разраб.		Черноок				ЭВМ, гр. 150501		
Пров.		Поденок						

ПРИЛОЖЕНИЕ Б
(обязательное)

Схема алгоритма функции `show_daemon_cond()`

Изм	Лист	№ документа	Подпись	Дата
Разраб.		Черноок		
Пров.		Поденок		

ГУИР.400201.124 ПД2

Схема алгоритма функции show_daemon_cond ()

Лит.	Масса	Масштаб
У		

Лист	Листов 1

ЭВМ, гр. 150501

ПРИЛОЖЕНИЕ В
(обязательное)

Схема алгоритма функции `daemon_create()`

Изм	Лист	№ документа	Подпись	Дата
Разраб.		Черноок		
Пров.		Поденок		

ГУИР.400201.124 ПД3

Схема алгоритма функции daemon_create ()

Лит.	Масса	Масштаб
У		

Лист	Листов 1

ЭВМ, гр. 150501

ПРИЛОЖЕНИЕ Г
(обязательное)

Схема алгоритма функции `file_digest_update()`

Изм	Лист	№ документа	Подпись	Дата
Разраб.		Черноок		
Пров.		Поденок		

ГУИР.400201.124 ПД4

Схема алгоритма функции file_digest_update ()

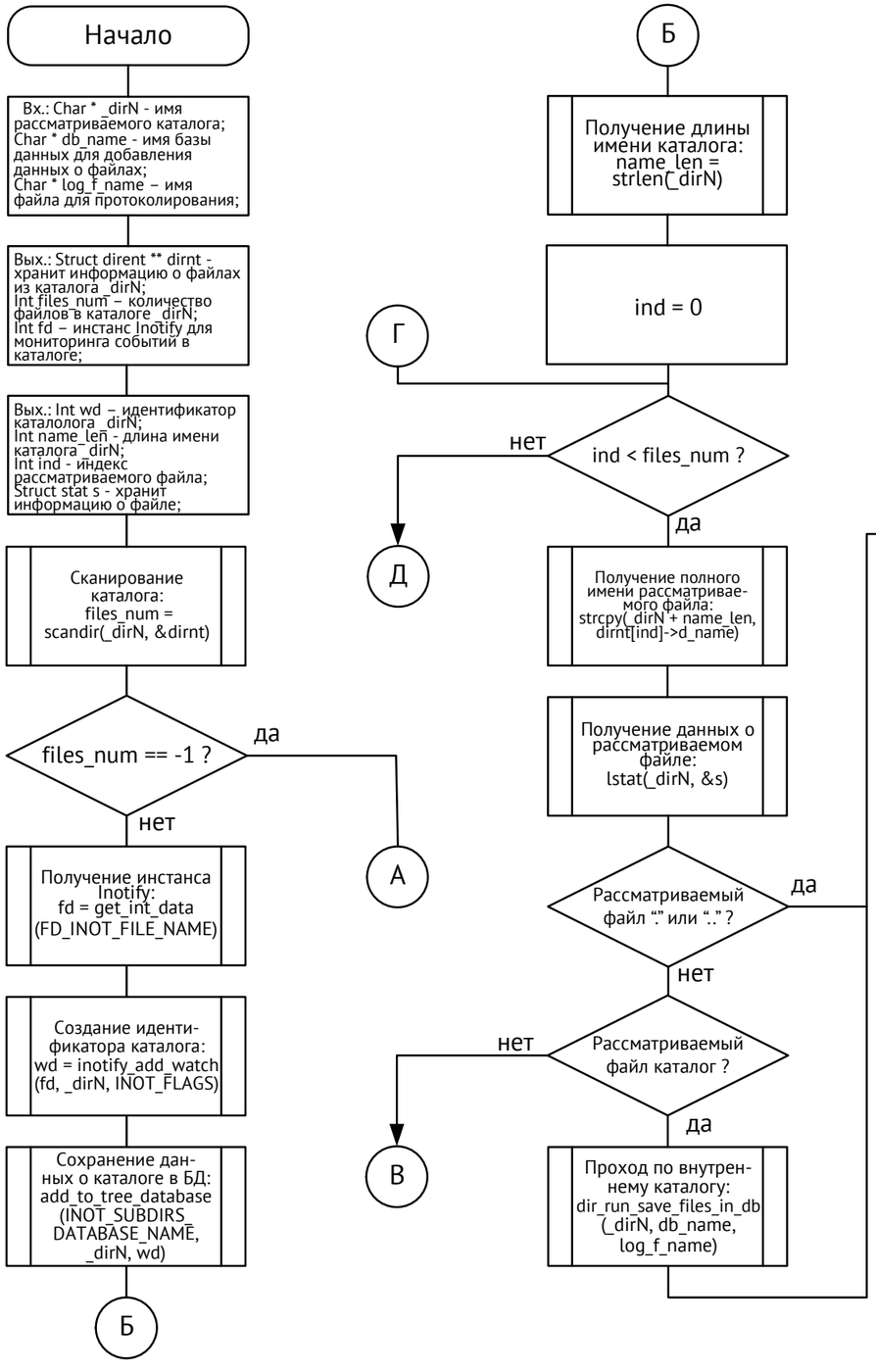
Лит.	Масса	Масштаб
У		

Лист	Листов 1

ЭВМ, гр. 150501

ПРИЛОЖЕНИЕ Д
(обязательное)

Схема алгоритма функции `dir_run_save_files_in_db()`



Изм	Лист	№ документа	Подпись	Дата	ГУИР.400201.124 ПД5			
Разраб.		Черноок			Схема алгоритма функции dir_run_save_files_in_db ()	Лит.	Масса	Масштаб
Пров.		Поденок				У		
						Лист	Листов 1	
						ЭВМ, гр. 150501		

ПРИЛОЖЕНИЕ Е
(обязательное)

Код программы

ПРИЛОЖЕНИЕ Ж
(обязательное)

Ведомость курсового проекта

Обозначение	Наименование	Примечание
	Текстовые документы	
БГУИР КП 1-40 02 01 124 ПЗ	Пояснительная записка	36 с.
БГУИР КП 1-40 02 01 124 CD	Код программы	
	Графические документы	
ГУИР.400201.124 ПД1	Схема алгоритма main()	Формат А3
ГУИР.400201.124 ПД2	Схема алгоритма show_daemon_cond()	Формат А4
ГУИР.400201.124 ПД3	Схема алгоритма daemon_create()	Формат А4
ГУИР.400201.124 ПД4	Схема алгоритма file_digest_update()	Формат А4
ГУИР.400201.124 ПД5	Схема алгоритма dir_run_save_files_in_db()	Формат А3

Изм	Лист	№ документа	Подпись	Дата	БГУИР КП 1-40 02 01 124 Д1		
Разраб.		Черноок			Утилита контроля появления дубликатов в файловой системе с заменой их на жесткие ссылки и протоколирования фактов замены	Лит.	Лист 36 / Листов 36
Пров.		Поденок				Т	
						ЭВМ, гр. 150501	