

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	5
1 ОБЗОР ЛИТЕРАТУРЫ.....	7
1.1 Анализ существующих аналогов.....	7
1.1.1 FileAudit (File and Folder Access Monitoring)	7
1.1.2 Linux Audit Daemon	8
1.1.3 Inotify-tools.....	9
1.2 Требования к работе программы	10
2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ	11
2.1 Блок интерфейса.....	12
2.2 Блок работы приложения	13
3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ	14
3.1 Functions	14
3.2 Mainwindow.....	16
4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ	18
4.1 Интерфейс пользователя	18
4.2 Файловая система.....	18
4.3 Блок конфигураций.....	18
4.4 Блок получения информации о файлах и каталогах	18
4.1.1 Переход по заданной директории	19
5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ.....	20
6 ТЕСТИРОВАНИЕ	23
ЗАКЛЮЧЕНИЕ	26
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	27
ПРИЛОЖЕНИЕ А	28
ПРИЛОЖЕНИЕ Б.....	29
ПРИЛОЖЕНИЕ В	30
ПРИЛОЖЕНИЕ Г.....	31
ПРИЛОЖЕНИЕ Д	39

ВВЕДЕНИЕ

Системный монитор – это аппаратный или программный компонент, используемый для мониторинга системных ресурсов и производительности в компьютерной системе.

Среди проблем управления, связанных с использованием инструментов системного мониторинга, – использование ресурсов, конфиденциальность и доступ к файлам.

Одним из важных преимуществ системного монитора является его возможность предоставления информации визуально, используя графики, диаграммы и таблицы, что позволяет пользователям более наглядно понимать текущее состояние системы. Он позволяет отслеживать использование ресурсов, таких как процессор, оперативная память, жесткий диск и сеть, что в свою очередь помогает выявить проблемы с производительностью и эффективно использовать ресурсы.

Системные мониторы можно классифицировать по таким критериям, как область мониторинга, режим работы системного монитора, платформа и операционная система, функциональность и уровень детализации.

Мониторы производительности отслеживают и измеряют производительность различных компонентов системы, таких как процессор, память и сеть. Они предоставляют информацию о загрузке ресурсов, времени отклика и других показателях производительности.

Мониторы ресурсов следят за использованием физических ресурсов компьютерной системы, таких как процессорное время, память, дисковое пространство и сетевая пропускная способность. Они помогают определить, насколько эффективно используются ресурсы.

Мониторы сети отслеживают сетевую активность, такую как пропускная способность, использование протоколов, пакеты данных и сетевые подключения. Они могут предоставлять информацию о потоках данных, задержках, ошибочных пакетах и других сетевых параметрах.

Мониторы приложений специализируются на отслеживании и анализе работы конкретных приложений или служб. Они могут предоставлять информацию о нагрузке, использовании ресурсов, производительности и ошибочных событиях, связанных с приложениями.

Мониторы событий и журналов записывают и анализируют события, происходящие в операционной системе и приложениях. Они помогают идентифицировать проблемы, ошибки, аварийные ситуации и отклонения от нормального поведения.

Мониторы безопасности следят за безопасностью системы, обнаруживают подозрительную активность, атаки или нарушения политик безопасности. Они могут записывать логи событий, анализировать сетевой трафик, сканировать на уязвимости и предоставлять информацию о статусе безопасности.

Мониторы доступа к файлам относятся к типу мониторов безопасности. Они обеспечивают контроль и аудит доступа к файлам в компьютерной системе и помогают предотвратить несанкционированный доступ к конфиденциальным данным, обнаружить подозрительную активность и обеспечить соответствие политикам безопасности. Также мониторы доступа к файлам могут предоставлять журналы событий, которые записывают информацию о доступе к файлам, изменениях, совершенных с файлами, и других операциях с файловой системой. Это помогает администраторам системы анализировать и аудитировать действия пользователей, выявлять потенциальные уязвимости и реагировать на инциденты безопасности.

Важно отметить, что мониторы доступа к файлам работают в тесной связи с политиками безопасности и требованиями конкретной системы. Они должны быть настроены и адаптированы к уникальным потребностям организации или пользователя. Кроме того, эффективное использование мониторов доступа к файлам требует постоянного анализа и обновления, чтобы обеспечить актуальность и соответствие требованиям безопасности.

Мониторы доступа к файлам играют важную роль в обеспечении безопасности и контроля доступа к файловой системе компьютерной системы. Они предоставляют гибкие и мощные механизмы для определения и управления доступом к файлам, а также аудита операций с файлами. Независимо от типа монитора доступа к файлам, их использование является важной частью общей стратегии безопасности информационной системы.

Безопасность информационной системы является критически важным аспектом для любой организации или пользователя. Она охватывает меры и практики, направленные на защиту информации от несанкционированного доступа, использования, раскрытия, изменения, уничтожения или нарушения ее целостности. Многие организации работают с конфиденциальной информацией, такой как персональные данные клиентов, финансовые данные, бизнес-секреты и интеллектуальная собственность. Нарушение конфиденциальности может привести к серьезным последствиям, включая утечку данных, нарушение законодательства о защите данных и утрату доверия со стороны клиентов и партнеров.

Обеспечение безопасности информационной системы – это непрерывный процесс, который требует планирования, реализации и постоянного мониторинга. Монитор доступа файлов является лишь одним из компонентов общей стратегии безопасности информационной системы. Он должен использоваться в сочетании с другими мерами безопасности, такими как авторизация и аутентификация, шифрование данных, брандмауэры и регулярные обновления системы. Вместе эти меры создают более надежную и защищенную информационную систему.

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Анализ существующих аналогов

Перед началом выполнения курсового проекта необходимо провести анализ уже существующих утилит системных мониторов доступа к файлам.

1.1.1 FileAudit (File and Folder Access Monitoring)

FileAudit (File and Folder Access Monitoring) – это программное обеспечение, специализирующееся на мониторинге доступа к файлам и папкам в операционных системах Windows. Он обеспечивает детальный аудит доступа к файлам, что позволяет организациям отслеживать и анализировать все операции, связанные с файлами, выполняемые пользователями и группами в сети.

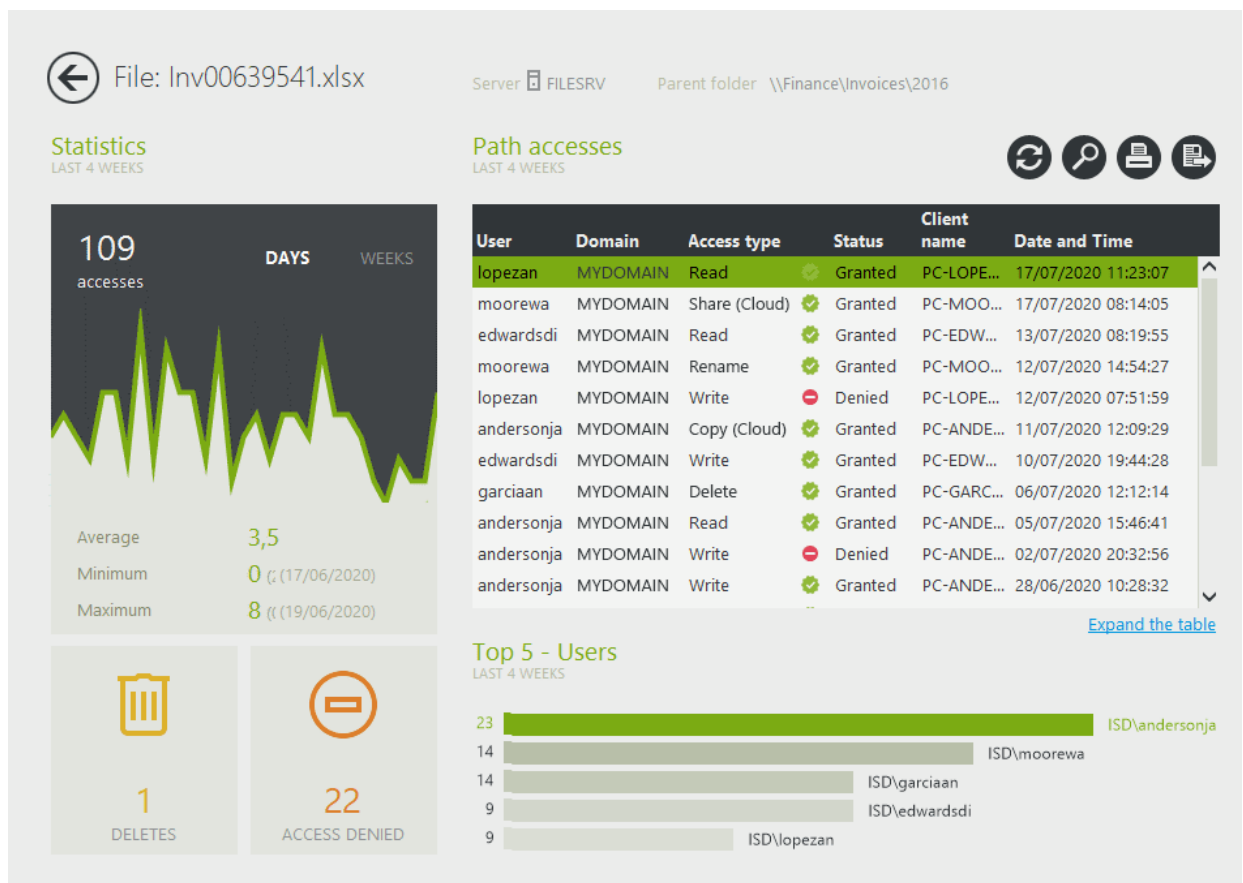


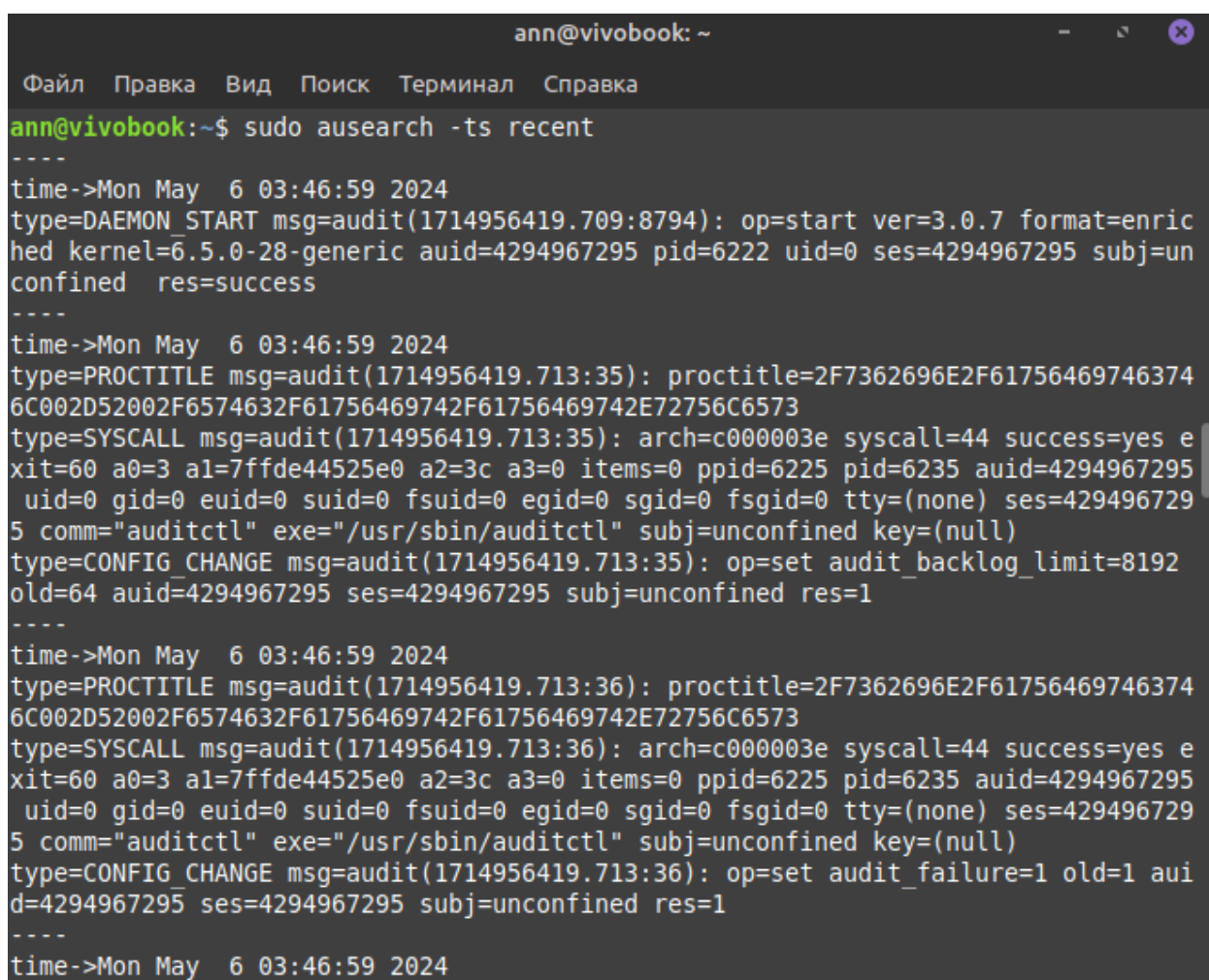
Рисунок 1.1 – Скриншот работы программы FileAudit

FileAudit записывает информацию о каждом действии, связанном с файлом или папкой, таком как чтение, запись, изменение, удаление, переименование и перемещение файлов. Он также отслеживает доступ через удаленные сетевые ресурсы.

FileAudit создает подробные журналы аудита, которые содержат информацию о действиях пользователя, времени и дате, IP-адресе, имени компьютера и других событийных данных. Журналы аудита могут быть сохранены в централизованном хранилище или отправлены на электронную почту или систему управления событиями.

1.1.2 Linux Audit Daemon

Linux Audit Daemon, также известный как `auditd`, является компонентом системы аудита в операционной системе Linux. Он позволяет аудитировать операции чтения, записи, исполнения и удаления файлов, а также изменения прав доступа к файлам и каталогам.



```
ann@vivobook: ~
Файл  Правка  Вид  Поиск  Терминал  Справка
ann@vivobook:~$ sudo ausearch -ts recent
----
time->Mon May 6 03:46:59 2024
type=DAEMON_START msg=audit(1714956419.709:8794): op=start ver=3.0.7 format=enriched kernel=6.5.0-28-generic auid=4294967295 pid=6222 uid=0 ses=4294967295 subj=unconfined res=success
----
time->Mon May 6 03:46:59 2024
type=PROCTITLE msg=audit(1714956419.713:35): proctitle=2F7362696E2F617564697463746C002D52002F6574632F61756469742F61756469742E72756C6573
type=SYSCALL msg=audit(1714956419.713:35): arch=c000003e syscall=44 success=yes exit=60 a0=3 a1=7ffde44525e0 a2=3c a3=0 items=0 ppid=6225 pid=6235 auid=4294967295 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=(none) ses=4294967295 comm="auditctl" exe="/usr/sbin/auditctl" subj=unconfined key=(null)
type=CONFIG_CHANGE msg=audit(1714956419.713:35): op=set audit_backlog_limit=8192 old=64 auid=4294967295 ses=4294967295 subj=unconfined res=1
----
time->Mon May 6 03:46:59 2024
type=PROCTITLE msg=audit(1714956419.713:36): proctitle=2F7362696E2F617564697463746C002D52002F6574632F61756469742F61756469742E72756C6573
type=SYSCALL msg=audit(1714956419.713:36): arch=c000003e syscall=44 success=yes exit=60 a0=3 a1=7ffde44525e0 a2=3c a3=0 items=0 ppid=6225 pid=6235 auid=4294967295 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=(none) ses=4294967295 comm="auditctl" exe="/usr/sbin/auditctl" subj=unconfined key=(null)
type=CONFIG_CHANGE msg=audit(1714956419.713:36): op=set audit_failure=1 old=1 auid=4294967295 ses=4294967295 subj=unconfined res=1
----
time->Mon May 6 03:46:59 2024
```

Рисунок 1.2 – Скриншот работы системы аудита `auditd`

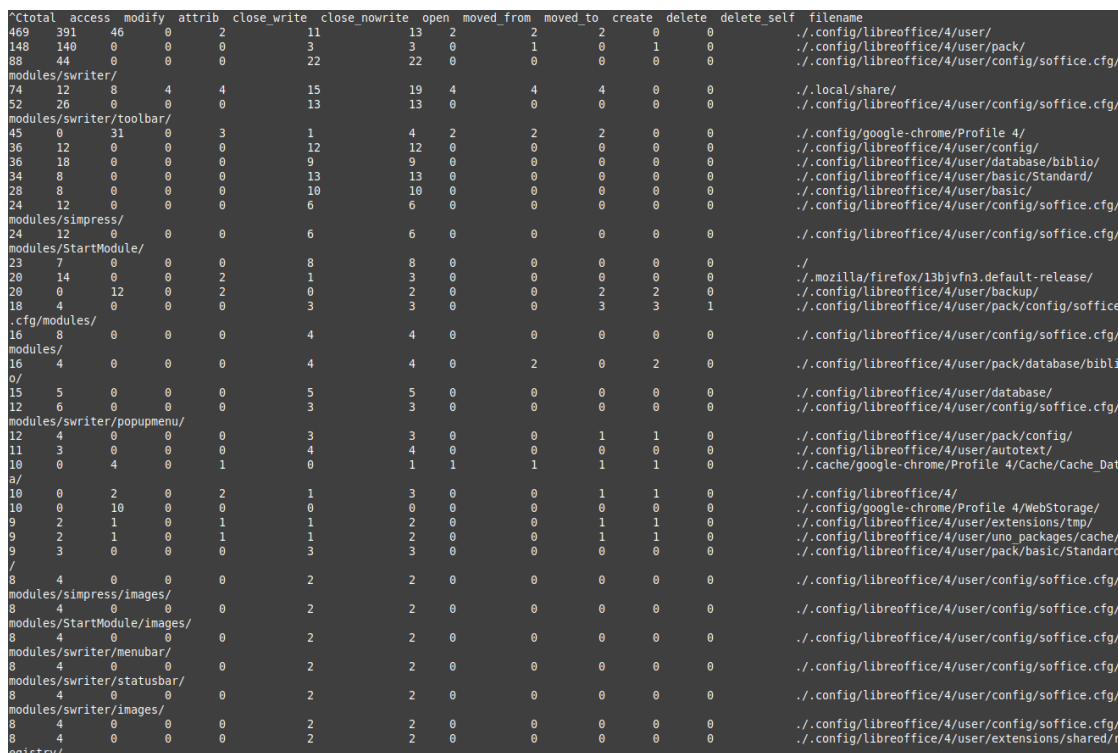
С помощью инструментов, таких как `auditctl`, можно создавать правила аудита, определяющие, какие события доступа к файлам должны быть аудиторваны. Например, можно настроить правило, чтобы аудитировать все операции чтения и записи файлов в конкретной директории или на определенных файловых системах.

Когда настроены соответствующие правила аудита, Linux Audit Daemon начинает регистрировать события доступа к файлам в соответствующих журналах аудита. Это включает информацию о пользователе, выполнившем операцию, времени и дате, типе операции (например, чтение или запись), пути к файлу и других метаданных.

Журналы аудита, созданные Linux Audit Daemon, могут быть анализированы для выявления несанкционированного доступа к файлам или других подозрительных действий. Администраторы могут использовать инструменты анализа журналов, такие как `ausearch` или `aureport`, для фильтрации и просмотра аудиторских записей, основываясь на различных критериях, таких как пользователь, файл, тип операции и временной диапазон. На рисунке 1.2 приведен пример использования команды `ausearch` используется для поиска и фильтрации записей аудита в системных журналах. Параметр `-ts recent` указывает на то, что нужно выполнить поиск событий, произошедших недавно. Она отображает последние записи аудита, что может быть полезно для мониторинга и анализа активности в системе.

1.1.3 Inotify-tools

Inotify-tools представляет собой набор утилит для мониторинга изменений в файловой системе Linux в реальном времени с использованием механизма `inotify` ядра Linux. Они предоставляют простой способ отслеживания событий, связанных с файлами и директориями, и выполняют действия при возникновении этих событий.



total	access	modify	attrib	close_write	close_nowrite	open	moved_from	moved_to	create	delete	delete_self	filename
469	391	46	0	2	11	13	2	2	2	0	0	./config/libreoffice/4/user/
148	140	0	0	0	3	3	0	1	0	1	0	./config/libreoffice/4/user/pack/
88	44	0	0	0	22	22	0	0	0	0	0	./config/libreoffice/4/user/config/soffice.cfg/
												modules/swriter/
74	12	8	4	4	15	19	4	4	4	0	0	./local/share/
52	26	0	0	0	13	13	0	0	0	0	0	./config/libreoffice/4/user/config/soffice.cfg/
												modules/swriter/toolbar/
45	0	31	0	3	1	4	2	2	2	0	0	./config/google-chrome/Profile 4/
36	12	0	0	0	12	12	0	0	0	0	0	./config/libreoffice/4/user/config/
36	18	0	0	0	9	9	0	0	0	0	0	./config/libreoffice/4/user/database/biblio/
34	0	0	0	0	13	13	0	0	0	0	0	./config/libreoffice/4/user/basic/Standard/
28	0	0	0	0	10	10	0	0	0	0	0	./config/libreoffice/4/user/basic/
24	12	0	0	0	6	6	0	0	0	0	0	./config/libreoffice/4/user/config/soffice.cfg/
												modules/simpress/
24	12	0	0	0	6	6	0	0	0	0	0	./config/libreoffice/4/user/config/soffice.cfg/
												modules/StartModule/
23	7	0	0	0	8	8	0	0	0	0	0	./
20	14	0	0	2	1	3	0	0	0	0	0	./mozilla/firefox/13bjvfn3.default-release/
20	0	12	0	2	0	2	0	0	2	2	0	./config/libreoffice/4/user/backup/
18	4	0	0	0	3	3	0	0	3	3	1	./config/libreoffice/4/user/pack/config/soffice
												.cfg/modules/
16	8	0	0	0	4	4	0	0	0	0	0	./config/libreoffice/4/user/config/soffice.cfg/
												modules/
16	4	0	0	0	4	4	0	2	0	2	0	./config/libreoffice/4/user/pack/database/bibli
												o/
15	5	0	0	0	5	5	0	0	0	0	0	./config/libreoffice/4/user/database/
12	6	0	0	0	3	3	0	0	0	0	0	./config/libreoffice/4/user/config/soffice.cfg/
												modules/swriter/popupmenu/
12	4	0	0	0	3	3	0	0	1	1	0	./config/libreoffice/4/user/pack/config/
11	3	0	0	0	4	4	0	0	0	0	0	./config/libreoffice/4/user/autotext/
10	0	4	0	1	0	1	1	1	1	1	0	./cache/google-chrome/Profile 4/Cache/Cache_Dat
												a/
10	0	2	0	2	1	3	0	0	1	1	0	./config/libreoffice/4/
10	0	10	0	0	0	0	0	0	0	0	0	./config/google-chrome/Profile 4/WebStorage/
9	2	1	0	1	1	2	0	0	1	1	0	./config/libreoffice/4/user/extensions/tmp/
9	2	1	0	1	1	2	0	0	1	1	0	./config/libreoffice/4/user/uno_packages/cache/
9	3	0	0	0	3	3	0	0	0	0	0	./config/libreoffice/4/user/pack/basic/Standard
												/
8	4	0	0	0	2	2	0	0	0	0	0	./config/libreoffice/4/user/config/soffice.cfg/
												modules/simpress/images/
8	4	0	0	0	2	2	0	0	0	0	0	./config/libreoffice/4/user/config/soffice.cfg/
												modules/StartModule/Images/
8	4	0	0	0	2	2	0	0	0	0	0	./config/libreoffice/4/user/config/soffice.cfg/
												modules/swriter/menubar/
8	4	0	0	0	2	2	0	0	0	0	0	./config/libreoffice/4/user/config/soffice.cfg/
												modules/swriter/statusbar/
8	4	0	0	0	2	2	0	0	0	0	0	./config/libreoffice/4/user/config/soffice.cfg/
												modules/swriter/images/
8	4	0	0	0	2	2	0	0	0	0	0	./config/libreoffice/4/user/config/soffice.cfg/
8	4	0	0	0	2	2	0	0	0	0	0	./config/libreoffice/4/user/extensions/shared/r
												egistry/

Рисунок 1.3 – Скриншот работы утилиты Inotify-tools

```

ann@vivobook:~$ inotifywatch -r /mnt/E/univer/kursovaya4sem
Establishing watches...
Finished establishing watches, now collecting statistics.
^Ctotal access modify close_write close_nowrite open create delete filename
349 182 99 10 21 31 3 3 /mnt/E/univer/kursovaya4sem/
149 53 34 12 16 27 3 4 /mnt/E/univer/kursovaya4sem/пояснительная записка/
5 3 0 0 1 1 0 0 /mnt/E/univer/kursovaya4sem/CourseWork4thTerm-master/
5 3 0 0 1 1 0 0 /mnt/E/univer/kursovaya4sem/CourseWork4thTerm-master (копия)/
ann@vivobook:~$

```

Рисунок 1.4 – Скриншот работы утилиты Inotify-tools для каталога

На рисунках 1.3 для всей файловой системы и рисунке 1.4 для каталога /mnt/E/univer/kyrsovaya4sem приведены примеры использования Inotifywatch – это утилита командной строки в Linux, которая позволяет отслеживать события файловой системы с использованием механизма inotify. Inotifywatch предоставляет информацию о событиях, происходящих в указанном каталоге или файле, и позволяет анализировать активность файловой системы.

1.2 Требования к работе программы

После рассмотрения аналогов можно сказать, что все они обладают большим количеством функций, которые невозможно реализовать в курсовом проекте за данный период времени. Поэтому были выбраны несколько ключевых возможностей, которые будут выполнены в рамках одного семестра:

- Программа должна иметь удобный пользовательский интерфейс;
- В программе должно быть реализовано сканирование файлов;
- В программе должен быть вывод параметров доступа к файлам.

В качестве языка программирования выбран язык программирования C++, который является языком низкого уровня и обеспечивает высокую производительность и эффективность. Это особенно важно для системного мониторинга, где требуется обработка больших объемов данных и быстрая реакция на изменения в файловой системе. C++ обеспечивает непосредственный доступ к системным функциям Linux через системные вызовы, что позволяет эффективно взаимодействовать с ядром операционной системы и получать информацию о доступе к файлам.

Для создания пользовательского интерфейса была выбран фреймворк Qt, который предоставляет обширный набор инструментов и компонентов для данных целей. Qt является кросс-платформенным фреймворком, который позволяет разрабатывать приложения, работающие на различных операционных системах, включая Linux, Windows и macOS. Это обеспечивает возможность создания монитора доступа к файлам, который может быть развернут на разных платформах без необходимости переписывания кода.

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

В GNU/Linux как и других Unix-подобных операционных системах понятие типа файла не связано с расширением файла (несколькими буквами после точки в конце имени), как это обстоит в Windows.

Unix-подобная ОС не следит за расширениями файлов. Задача связать расширения файла с конкретным пользовательским приложением, в котором этот файл будет открываться, видимо лежит на какой-либо дополнительной программе. В свою очередь пользовательское приложение анализирует структуру данных файла, расширение ему также безразлично.

Таким образом, среди файловых атрибутов, хранящихся в операционной системе на базе ядра Linux, нет информации о типе данных в файле. Там есть информация о более существенном разделении, связанном с тем, что в Unix-подобных системах все объекты – это файлы. Все объекты весьма разнообразны. Поэтому тип файла в Linux – это скорее тип объекта, но не тип данных как в Windows.

В операционной системе GNU/Linux существуют следующие типы файлов: обычные файлы, каталоги, символьные ссылки, блочные устройства, символьные устройства, сокеты, каналы. Каждый тип имеет собственное обозначение одним символом.

После определения требований к функционалу разрабатываемого приложения его следует разбить на функциональные блоки. Такой подход упростит понимание проекта, позволит устранить проблемы в архитектуре, обеспечит гибкость и масштабируемость программного продукта в будущем путем добавления новых блоков.

Для начала следует понять, какие функции должен выполнять наш системный монитор. В первую очередь, конечно же, должен выводиться весь список файлов, и папок, содержащихся в корневой директории. Более того, исходя из названия темы следует, должны выводиться права доступа для каждого файла, а именно доступ к чтению, записи в файл, запуска данной программы. Также требуется функционал, позволяющий переходить по всем папкам. Для более удобного пользования программой, требуется реализовать возможность перехода в предыдущую папку и возврат в корневую папку.

В операционной системе Linux есть много отличных функций безопасности, но она из самых важных – это система прав доступа к файлам. Каждый файл имеет три категории доступа: владелец, группа и остальные.

Владелец – набор прав для владельца файла, пользователя, который его создал или сейчас установлен его владельцем. Обычно владелец имеет все права, чтение, запись и выполнение.

Группа – любая группа пользователей, существующая в системе и привязанная к файлу. Но это может быть только одна группа и обычно это группа владельца, хотя для файла можно назначить и другую группу.

Остальные – все пользователи, кроме владельца и пользователей, входящих в группу файла.

Каждый файл имеет три параметра доступа. Чтение – разрешает получать содержимое файла, но на запись нет. Для каталога позволяет получить список файлов и каталогов, расположенных в нем. Запись – разрешение на запись новых данных в файл или изменение существующих, а также возможность создавать и изменять файлы и каталоги; Выполнение – нельзя выполнить программу, если у нее нет флага выполнения. Этот атрибут устанавливается для всех программ и скриптов, именно с помощью него система может понять, что этот файл нужно запускать как программу.

2.1 Блок интерфейса

Блок пользовательского интерфейса является важной частью разрабатываемого проекта и предназначен для обеспечения взаимодействия пользователя с приложением. Он обеспечивает представление данных и их визуализацию, позволяя пользователю управлять функциональностью приложения с помощью мыши и клавиатуры.

Для создания простого пользовательского интерфейса в данном проекте используется фреймворк Qt. Qt – это мощный инструментальный набор разработчиков приложений с графическим интерфейсом, который предоставляет широкий набор компонентов и инструментов для создания интерактивных пользовательских интерфейсов.

Пользовательский интерфейс включает в себя набор пунктов меню, которые предоставляют пользователю доступ к различным функциям и операциям приложения. Пользователь может взаимодействовать с этими пунктами меню, выбирая их с помощью мыши или клавиатуры, что инициирует выполнение соответствующих действий или операций.

Создание простого пользовательского интерфейса с помощью Qt обычно включает в себя создание объектов классов, предоставляемых фреймворком, таких как QMainWindow, QMenu, QAction и других. Эти объекты используются для создания основного окна приложения, меню, подменю и действий, доступных пользователю.

В данном проекте пользовательский интерфейс будет взаимодействовать с другими компонентами приложения, такими как обработчики событий и функции для работы с файловой системой. Пользовательский интерфейс будет отображать данные о директориях, файлах и правах доступа в виде таблицы, а также обновляться при взаимодействии пользователя с приложением.

В результате создания пользовательского интерфейса с использованием Qt пользователь получает удобный и интуитивно понятный способ взаимодействия с приложением, а разработчики получают мощный инструментальный набор для создания интерактивных и кросс-платформенных приложений.

2.2 Блок работы приложения

Блок работы приложения необходим для взаимодействия пользователя с приложением, а также обработки сигналов других блоков и является неотъемлемой его частью. Тут осуществляется весь необходимый функционал и возможности:

- Выбор директории: пользователь имеет возможность выбрать директорию, с которой будет производиться сканирование файлов и контроль доступ к ним;

- Просмотр обработанных файлов в режиме реального времени: приложение обеспечивает возможность просмотра обработанных файлов в режиме реального времени. Это означает, что по мере обнаружения и обработки новых файлов, они будут отображаться пользователю немедленно, без необходимости перезагрузки приложения;

- Вывод параметров доступа к файлам для текущего пользователя: Блок работы приложения отвечает за получение и отображение параметров доступа к файлам для текущего пользователя. Это может включать информацию о правах доступа, таких как чтение, запись и выполнение, а также информацию о владельцах и группах пользователей, имеющих доступ к файлам.

Блок сканирования файлов предназначен для прохода по всем директориям и поддиректориям, начиная с заданной пользователем директории. Он выполняет сканирование файловой системы и заносит найденные файлы в массив для дальнейшего использования. Это позволяет приложению получить полный список файлов, с которыми будет работать, и обеспечить доступ к ним в дальнейшем.

Блок сканирования файлов может использовать различные алгоритмы и методы для прохода по директориям и поиска файлов. Например, он может использовать рекурсивный алгоритм, чтобы пройти по всем поддиректориям, или использовать итеративный алгоритм с использованием стека или очереди для обхода файловой системы.

В результате работы блока сканирования файлов приложение получает полный список файлов для дальнейшего анализа и контроля доступа к ним. Это обеспечивает возможность более эффективного и удобного управления файлами и их доступом в рамках приложения.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Функциональное проектирование является одним из важных этапов разработки программного обеспечения. В данном разделе будет определена и описана функциональность системы монитора доступа к файлам, ее основные компоненты и взаимодействие между ними.

3.1 Functions

– `string startPage`: эта функция является точкой входа программы. Она принимает ссылки на различные массивы и переменные, такие как предыдущая директория, указатель на массив содержащихся директорий, указатель на массив содержащихся файлов, указатель на массив соответствующих режимов доступа, количество директорий, количество файлов, количество соответствующих режимов доступа, используемые для хранения информации о директориях, файлах и правах доступа. Функция сначала инициализирует переменную `previousDirectory` значением домашней директории текущего пользователя. Это реализуется путем вызова функции `getuid()`, которая возвращает идентификатор пользователя (User ID, UID) текущего процесса. UID – это уникальный числовой идентификатор, присвоенный каждому пользователю в Linux. Функция `getuid()` возвращает этот UID, который затем сохраняется в переменной `user_id` типа `uid_t`. Затем вызывается функция `getpwuid()`, которая используется для получения информации о пользователе по его UID. Она принимает UID в качестве аргумента и возвращает структуру `passwd`, содержащую информацию о пользователе, включая его имя пользователя (`login`), домашний каталог и другие данные. В данном коде результат функции сохраняется в указатель `user`, в предварительно объявленную переменную типа `struct passwd* user`. Затем происходит вызов функции `directoryBrowsing`, аргументами которой являются директория для исследования, а также массивы и переменные для заполнения информацией о директориях, файлах и правах доступа. Наконец, функция возвращает значение `previousDirectory`.

– `char* displayPermission`: Эта функция принимает значение `st_mode`, которое представляет режим доступа файла, и возвращает строку, представляющую права доступа к файлу. Она использует битовые операции для определения наличия или отсутствия определенных прав доступа (чтение, запись, выполнение) и формирует строку из символов 'r', 'w' и 'x'. Если файл имеет установленные флаги `S_ISUID`, `S_ISGID` или `S_ISVTX`, функция также добавляет символы 's' или 't' в соответствующие позиции строки. Результат возвращается в виде указателя на символьный массив.

`S_ISUID`, `S_ISGID` и `S_ISVTX` являются макросами, используемыми для проверки различных флагов доступа в поле `st_mode` структуры `stat`. Макрос `S_ISUID` используется для проверки флага `setuid` (установка идентификатора

пользователя). Флаг `setuid` применяется к исполняемым файлам и позволяет запустить программу с привилегиями пользователя-владельца файла вместо привилегий текущего пользователя. Если у файла установлен флаг `setuid`, то `S_ISUID` вернет ненулевое значение, иначе вернет ноль. Макрос `S_ISGID` используется для проверки флага `setgid` (установка идентификатора группы). Флаг `setgid` применяется к исполняемым файлам и директориям. Если у файла установлен флаг `setgid`, то `S_ISGID` вернет ненулевое значение, иначе вернет ноль. В случае директории с установленным флагом `setgid`, новые файлы, созданные в этой директории, будут иметь группу-владельца, совпадающую с группой-владельцем директории. Макрос `S_ISVTX` используется для проверки флага `sticky` (прилипчивый бит). Флаг `sticky` применяется к директориям и ограничивает возможность удаления файлов из этой директории другими пользователями. Если у директории установлен флаг `sticky`, то `S_ISVTX` вернет ненулевое значение, иначе вернет ноль.

– `unsigned directoryBrowsing`: Эта функция осуществляет обход директории и собирает информацию о содержащихся в ней файлах и поддиректориях. В качестве параметров она принимает `introducedDir` – строку, содержащую путь к директории, которую необходимо обойти, `directoriesArray` – указатель на массив строк, в котором будут храниться найденные поддиректории, `filesArray` – указатель на массив строк, в котором будут храниться найденные файлы, `permissionArray` – указатель на массив строк, в котором будут храниться информация о правах доступа для каждого файла/директории, `numberOfDirectories` – ссылка на переменную, в которой будет храниться количество найденных поддиректорий, `numberOfFiles` – ссылка на переменную, в которой будет храниться количество найденных файлов, `numberOfPermission` – ссылка на переменную, в которой будет храниться количество найденных прав доступа. Функция открывает директорию с помощью `opendir`, и если открытие директории не удалось (получен указатель `NULL`), функция возвращает 0. Затем происходит цикл, в котором для каждого элемента содержимого директории вызывается функция `lstat`, которая возвращает информацию о файле/директории. Если вызов `lstat` прошел успешно, происходит проверка типа элемента с помощью макросов `S_ISDIR`, `S_ISREG` и `S_ISLNK` (для директории, обычного файла и символической ссылки соответственно). Если элемент является директорией, информация о ней добавляется в массив `directoriesArray`, а также информация о правах доступа добавляется в массив `permissionArray`. Для обновления массивов `directoriesArray` и `permissionArray` выделяется новая память большего размера и копируются данные из старых массивов. Затем увеличивается счетчик `numberOfDirectories` и `numberOfPermission`. Если элемент является обычным файлом, он добавляется в массив `filesArray` и информация о правах доступа добавляется в массив `permissionArray`. Процесс обновления массивов `filesArray` и `permissionArray` аналогичен предыдущему случаю.

Если элемент является символической ссылкой, то ее информация добавляется только в массив `filesArray`. Если вызов `lstat` вернул ошибку, выводится сообщение об ошибке. После завершения цикла содержимое директории полностью просмотрено, и функция закрывает директорию с помощью `closedir`. В конце функция возвращает 0.

3.2 Mainwindow

– `MainWindow`: Это конструктор класса `MainWindow`, который наследуется от `QMainWindow`. Он принимает указатель `parent` на объект родительского класса `QWidget`. Затем вызывается конструктор родительского класса `QMainWindow`, чтобы выполнить инициализацию окна. Затем создается объект `ui` типа `Ui::MainWindow`, который представляет собой пользовательский интерфейс для окна. Метод `setupUi(this)` инициализирует пользовательский интерфейс для окна `MainWindow`. Он создает и размещает все виджеты, определенные в файле интерфейса `ui`, на окне. Далее создается динамический массив строк `directoriesArray` размером 1 элемент. Этот массив будет использоваться для хранения найденных поддиректорий. Создается динамический массив строк `filesArray` размером 1 элемент. Этот массив будет использоваться для хранения найденных файлов. вызывается функция `startPage`, которая принимает несколько параметров, включая `previousDirectory`, `directoriesArray`, `filesArray`, `permissionArray`, `numberOfDirectories`, `numberOfFiles` и `numberOfPermission`. Результат выполнения функции присваивается переменной `previousDirectory`. значение переменной `previousDirectory` присваивается переменной `currentDirectory`. вызывается функция `outputData`, которая принимает объект `ui` в качестве параметра. Эта функция отображает данные на пользовательском интерфейсе, используя объект `ui`.

– `void outputData`: Функция отвечает за вывод данных в графический интерфейс пользовательского приложения, используя объект `ui` типа `Ui::MainWindow`. Устанавливается количество столбцов в виджете таблицы `tableWidget` равным 2. Устанавливается количество строк в виджете таблицы `tableWidget` равным сумме `numberOfDirectories` и `numberOfFiles`. Устанавливается ширина столбцов в таблице `tableWidget`. Первый столбец имеет ширину 1000 пикселей, а второй столбец – 400 пикселей. Затем следуют два цикла `for`, которые используются для установки пустых значений в ячейки таблицы. Первый цикл устанавливает пустые значения в столбце 0 (первый столбец), а второй цикл – в столбце 1 (второй столбец). Количество итераций в обоих циклах равно `numberOfDirectories + numberOfFiles - 1`. В каждой итерации создается новый объект `QTableWidgetItem`, устанавливается пустое значение текста с помощью метода `setText(" ")`, и затем этот объект добавляется в таблицу с помощью метода `ui->tableWidget->setItem(ridx, colidx, item)`, где `ridx` – индекс строки, `colidx` – индекс столбца.

Следующий цикл `for` используется для заполнения первого столбца таблицы данными из массива `directoriesArray`. В каждой итерации преобразуется строка из `directoriesArray` в объект `QString` с помощью `QString::fromStdString(directoriesArray[ridx])`. Затем создается новый объект `QTableWidgetItem`, устанавливается текст, полученный из `QString`, и добавляется в таблицу в соответствующую ячейку. Затем идет цикл `for`, который заполняет первый столбец таблицы данными из массива `filesArray`. В каждой итерации преобразуется строка из `filesArray` в объект `QString`, а затем создается новый объект `QTableWidgetItem`, устанавливается текст и добавляется в таблицу. Последний цикл `for` заполняет второй столбец таблицы данными из массива `permissionArray`. Аналогично предыдущим циклам, каждый элемент из `permissionArray` преобразуется в объект `QString`, создается новый объект `QTableWidgetItem`, устанавливается текст и добавляется в таблицу. Устанавливается текст в элементе `lineEdit_2` (поле ввода) равным значению `currentDirectory`, преобразованному в объект `QString`. Очищается текст в элементе `lineEdit` (еще одно поле ввода). Зависит от условия `if (depthDir == 0)` устанавливает состояние кнопки `knopka`. Если `depthDir` равно 0, кнопка `knopka` отключается (`disabled`), в противном случае она включается (`enabled`), что является логичным, так как при нулевой глубине в иерархии каталогов нельзя вернуться на один коренной каталог выше.

- `~MainWindow`: Это деструктор класса `MainWindow`, который освобождает память, выделенную для пользовательского интерфейса.

- `void on_pushButton_clicked`: Эта функция вызывается при нажатии кнопки `pushButton`. Она получает текст из поля ввода `lineEdit` и преобразует его в строку. Если строка пустая, функция просто возвращает управление. Затем функция увеличивает значение `depthDir` на 1, формирует полный путь к директории и вызывает функцию `directoryBrowsing` для обработки новой директории. После этого функция вызывает `outputData` для обновления таблицы.

- `void on_knopka_clicked`: Эта функция вызывается при нажатии кнопки `knopka`. Она уменьшает значение `depthDir` на 1, обновляет текущую директорию и вызывает функцию `directoryBrowsing` для обработки новой директории. Затем она вызывает `outputData` для обновления таблицы.

- `void on_tableWidget_cellClicked`: Эта функция вызывается при щелчке по ячейке таблицы `tableWidget`. Она проверяет, щелкнули ли во второй столбец или на строку, соответствующую файлу (не директории). Если условие выполняется, функция просто возвращает управление. В противном случае она увеличивает значение `depthDir` на 1, получает текст из ячейки, соответствующей выбранной директории, и обновляет текущую директорию. Затем она вызывает `directoryBrowsing` для обработки новой директории и `outputData` для обновления таблицы.

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

В данном разделе описывается функционирование и структура разрабатываемого приложения для системного монитора файлов. Структурная схема программы и диаграмма последовательности приведены в Приложениях А и Б соответственно.

4.1 Интерфейс пользователя

Этот блок отвечает за отображение информации о файлах и каталогах в понятном для пользователя виде. Он показывает такие данные, как имена файлов и каталогов и их права доступа.

Интерфейс пользователя предоставляет инструменты для навигации по файловой системе, позволяя пользователю просматривать содержимое каталогов, перемещаться между ними. Блок принимает запросы от пользователя и передает их в другие блоки приложения.

4.2 Файловая система

Данный блок отвечает за взаимодействие с файловой системой операционной системы, предоставляя возможность получать информацию о файлах и каталогах.

Файловая система обеспечивает выполнение операций над файлами и каталогами по запросам, полученным от интерфейса пользователя.

4.3 Блок конфигураций

Этот блок отвечает за загрузку, хранение и применение текущих состояний приложения.

Он может содержать информацию о текущем рабочем каталоге, настройках доступа и другие параметры конфигурации.

Данные из блока конфигураций используются другими компонентами приложения для корректной работы.

4.4 Блок получения информации о файлах и каталогах

Данный блок отвечает за сбор и предоставление информации о файлах и каталогах, находящихся в файловой системе.

Он взаимодействует с файловой системой для получения данных о структуре файлов и каталогов.

Полученная информация передается в интерфейс пользователя для отображения и дальнейшего взаимодействия.

В данном подразделе будет рассмотрен алгоритм, используемый для перехода в заданную директорию и отображения всей информации.

4.1.1 Переход по заданной директории

Структурная схема программы представлена в Приложении В.

Шаг 1. Начало алгоритма.

Шаг 2. Объявление и инициализация необходимых переменных.

Шаг 3. Открытие потока каталога.

Шаг 4. Проверка на существовании директории, переданной в параметре функции.

Шаг 5. Если выражение (`dir == NULL`) истинно, вывести сообщение об ошибке, завершить программу.

Шаг 6. Если выражение (`dir == NULL`) ложно, продолжить выполнение программы.

Шаг 7. Чтение оглавление каталога `dir`.

Шаг 8. Начало цикла.

Шаг 9. Объявление структуры `stat`.

Шаг 10. Проверка на чтение из головной директории.

Шаг 11. Если выражение истинно, тогда чтение следующего оглавления каталога `dir`.

Шаг 12. Если выражение ложно, продолжить выполнение программы.

Шаг 13. Формирование полной директории к файлу.

Шаг 14. Проверка на получение информации о файле.

Шаг 15. Если выражение ложно, выводится ошибка, чтение следующего оглавления каталога.

Шаг 16. Если выражение истинно, проверяется тип файла.

Шаг 17. Если это папка, занести директорию в массив `directoriesArray`.

Шаг 18. Занести права доступа к папке в массив `permissionArray`.

Шаг 19. Если это обычный файл, занести директорию в массив `filesArray`.

Шаг 20. Занести права доступа к файлу в массив `permissionArray`.

Шаг 21. Если это ссылка, занести директорию в массив `filesArray`.

Шаг 22. Занести права доступа к ссылке в массив `permissionArray`.

Шаг 23. Чтение оглавления каталога.

Шаг 24. Проверка условия цикла.

Шаг 25. Если директория прочитана не полностью перейти на Шаг 9.

Шаг 26. Если директория прочитана полностью, выйти из цикла.

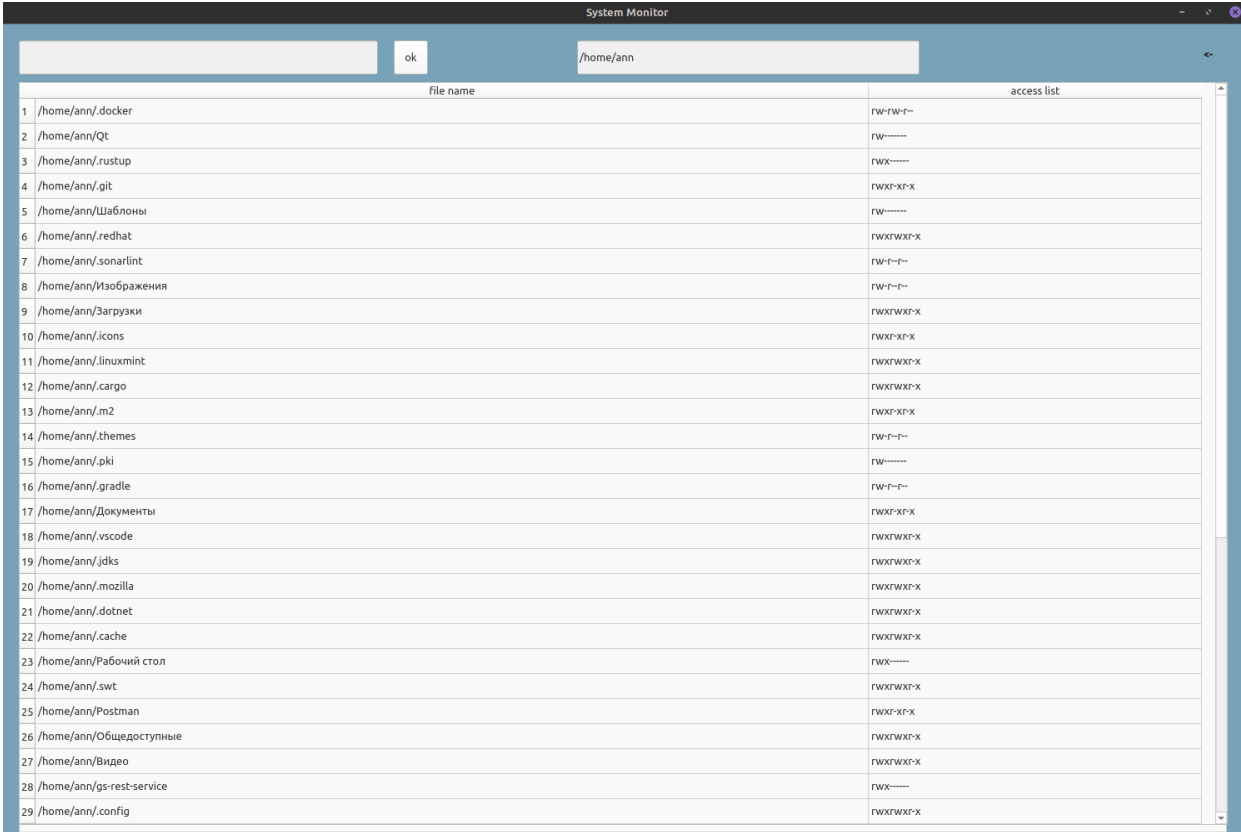
Шаг 28. Закрытие потока каталога.

Шаг 29. Конец алгоритма.

5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Требования для пользования программой: ОС – Linux, наличие QT на компьютере.

Функционал программы: данное приложение создано для вывода доступа к файлам всей файловой системы. В приложении создан удобный и интуитивный интерфейс. В программе можно перемещаться по всем папкам, выбирать интересующую категорию. На рисунке 5.1 представлен общий вид приложения.



The screenshot shows a window titled 'System Monitor' with a search bar at the top containing '/home/ann'. Below the search bar is a table with two columns: 'file name' and 'access list'. The table lists 29 files and their corresponding permissions.

	file name	access list
1	/home/ann/.docker	rw-rw-r--
2	/home/ann/.Qt	rw-----
3	/home/ann/.rustup	rwX-----
4	/home/ann/.git	rwXr-xr-x
5	/home/ann/Шаблоны	rw-----
6	/home/ann/.redhat	rwXrwxr-x
7	/home/ann/.sonarlint	rw-r--r--
8	/home/ann/Изображения	rw-r--r--
9	/home/ann/Загрузки	rwXrwxr-x
10	/home/ann/.icons	rwXr-xr-x
11	/home/ann/.linuxmint	rwXrwxr-x
12	/home/ann/.cargo	rwXrwxr-x
13	/home/ann/.m2	rwXr-xr-x
14	/home/ann/.themes	rw-r--r--
15	/home/ann/.pki	rw-----
16	/home/ann/.gradle	rw-r--r--
17	/home/ann/Документы	rwXr-xr-x
18	/home/ann/.vscode	rwXrwxr-x
19	/home/ann/.jdk	rwXrwxr-x
20	/home/ann/.mozilla	rwXrwxr-x
21	/home/ann/.dotnet	rwXrwxr-x
22	/home/ann/.cache	rwXrwxr-x
23	/home/ann/Рабочий стол	rwX-----
24	/home/ann/.swt	rwXrwxr-x
25	/home/ann/.Postman	rwXr-xr-x
26	/home/ann/Общедоступные	rwXrwxr-x
27	/home/ann/Видео	rwXrwxr-x
28	/home/ann/.gs-rest-service	rwX-----
29	/home/ann/.config	rwXrwxr-x

Рисунок 5.1 – Общий вид приложения

В правом столбце отображаются права доступа для разных пользователей, в левом – список файлов в домашней папке (начальная директория).

Первые 3 символа обозначают права доступа для владельца данного файла. Обозначения являются стандартными для операционной системы Linux. Символ «r» показывает, имеет ли пользователь доступ для чтения данного файла, символ «w» отвечает за возможность изменения файла и наконец символ «x» показывает, имеет ли пользователь возможность запускать данный файл. Следующие три символа показывают те же самые права доступа, однако не для владельца, а для группы. И последние 3 символа показывают доступ к файлу для всех остальных пользователей.

Для перехода по заданной директории необходимо в поле поиска написать продолжение пути относительно текущего и нажать кнопку «ок». Пример приведен на рисунке 5.2.

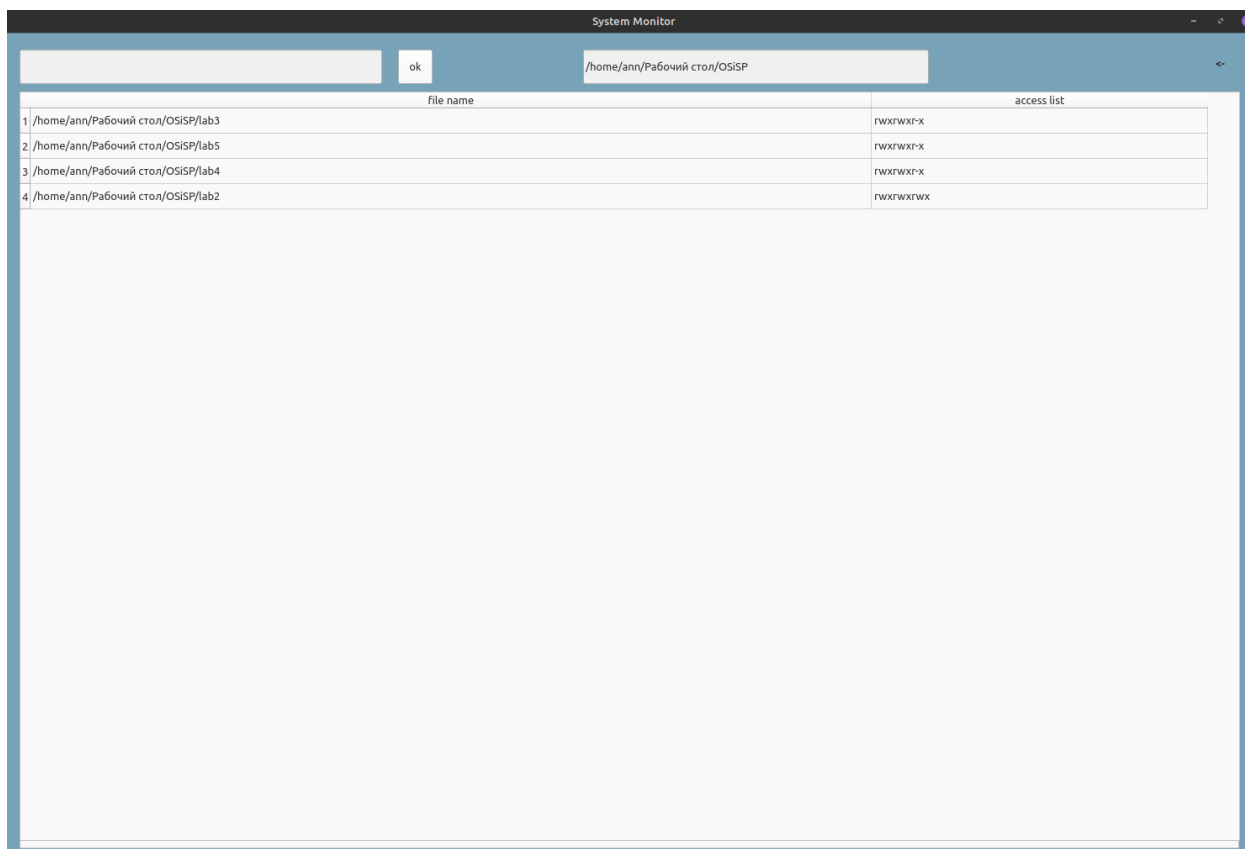


Рисунок 5.2 – Переход по заданной директории через поле поиска

Для того, чтобы вернуться на один уровень выше в корневую директорию, реализована кнопка «<-», которая становится доступна в момент, когда глубина директории становится не нулевой. Вид активной кнопки при наведении отображен на рисунке 5.3.

Более того, в программе реализована возможность перехода по любой директории из текущей директории. Пример приведен на рисунке 5.4.

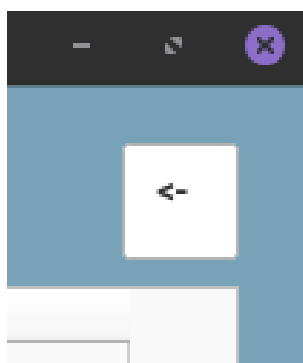


Рисунок 5.3 – Кнопка возврата в корневую директорию

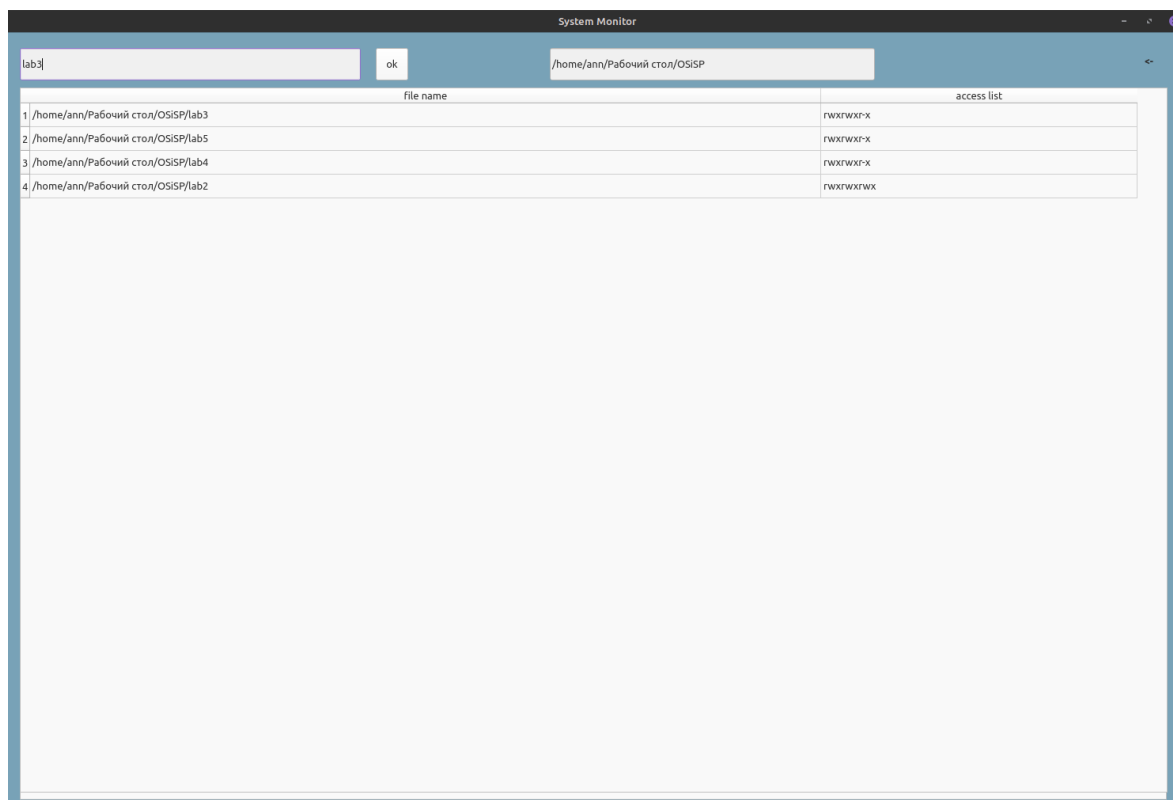


Рисунок 5.4 – Выбор директории, по которой можно перейти

Так же реализована возможность выбора папки из списка простым нажатием левой клавиши мыши, пример на рисунке 5.5.

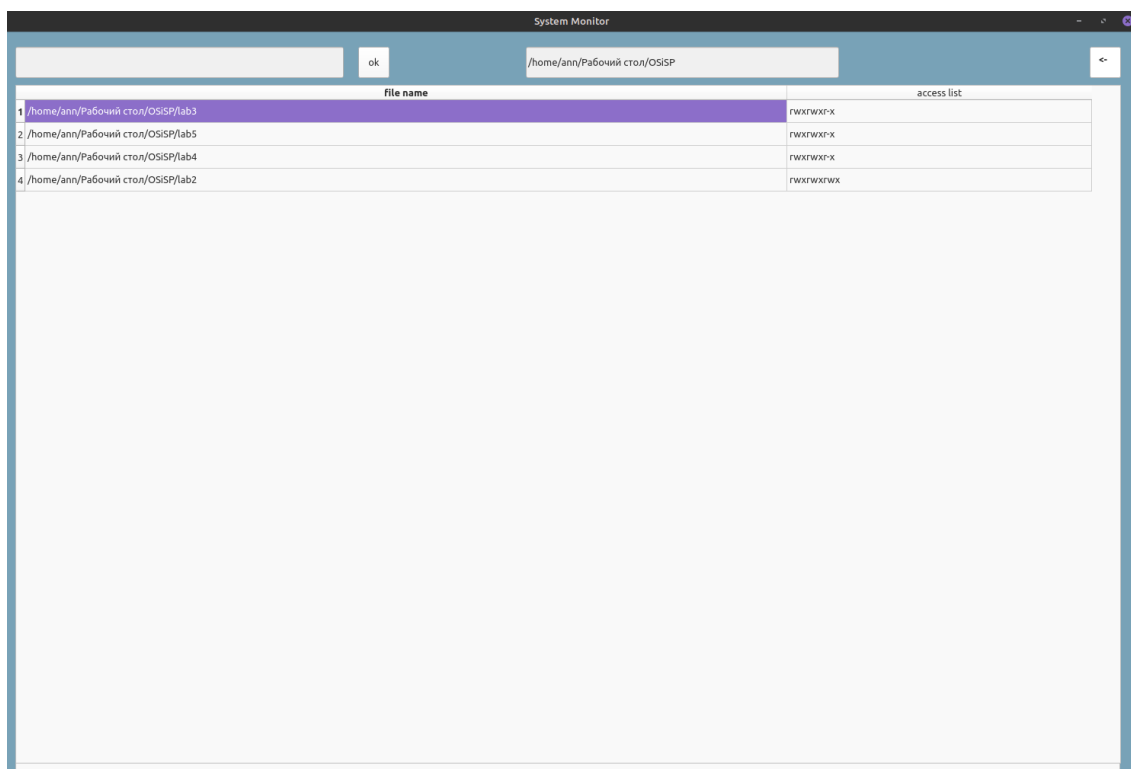
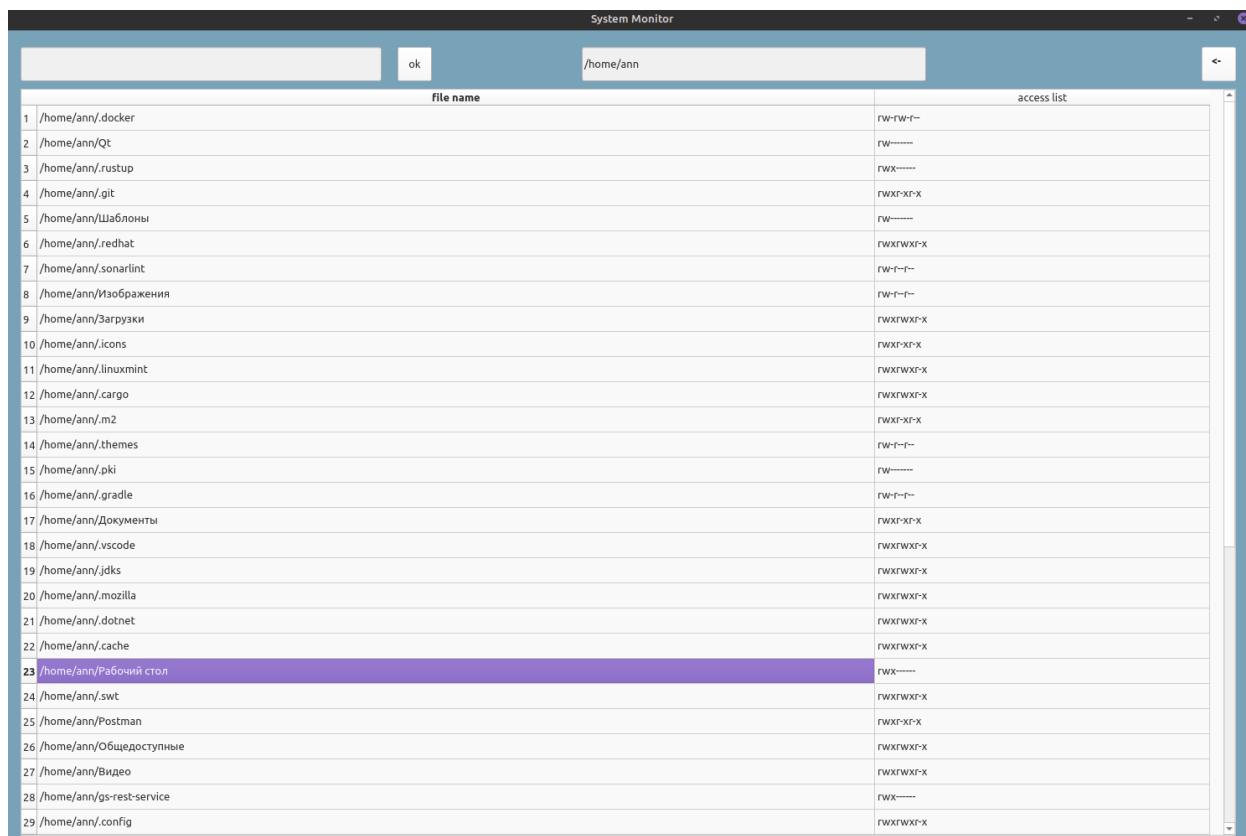


Рисунок 5.5 – Выбор папки из списка при помощи мыши

6 ТЕСТИРОВАНИЕ

Для начала, следует протестировать правильность работы программы. В первую очередь попробуем перейти по выбранной папке из общего списка директорий.

После запуска программы сразу отображается содержимое корневой директории, это отображено на рисунке 6.1. В качестве примера выберем папку «Рабочий стол».



	file name	access list
1	/home/ann/.docker	rw-rw-r--
2	/home/ann/.Qt	rw-----
3	/home/ann/.rustup	rwX-----
4	/home/ann/.git	rwXr-xr-x
5	/home/ann/Шаблоны	rw-----
6	/home/ann/.redhat	rwXrwXr-x
7	/home/ann/.sonarlint	rw-r--r--
8	/home/ann/Изображения	rw-r--r--
9	/home/ann/Загрузки	rwXrwXr-x
10	/home/ann/.icons	rwXr-xr-x
11	/home/ann/.linuxmint	rwXrwXr-x
12	/home/ann/.cargo	rwXrwXr-x
13	/home/ann/.m2	rwXr-xr-x
14	/home/ann/.themes	rw-r--r--
15	/home/ann/.pki	rw-----
16	/home/ann/.gradle	rw-r--r--
17	/home/ann/Документы	rwXr-xr-x
18	/home/ann/.vscode	rwXrwXr-x
19	/home/ann/.jdk	rwXrwXr-x
20	/home/ann/.mozilla	rwXrwXr-x
21	/home/ann/.dotnet	rwXrwXr-x
22	/home/ann/.cache	rwXrwXr-x
23	/home/ann/Рабочий стол	rwX-----
24	/home/ann/.swt	rwXrwXr-x
25	/home/ann/.Postman	rwXr-xr-x
26	/home/ann/Общедоступные	rwXrwXr-x
27	/home/ann/Видео	rwXrwXr-x
28	/home/ann/gs-rest-service	rwX-----
29	/home/ann/.config	rwXrwXr-x

Рисунок 6.1 – Тестовый переход по выбранной папке

После выбора папки «Рабочий стол» и нажатия на нее кнопкой мыши на экране видим ее содержимое. Это показано на рисунке 6.2

Следовательно, данный функционал является тоже полностью рабочим. Далее, протестируем функцию поиска папки по заданному имени. Введем в поле поиска название папки «OSiSP» и нажмем кнопку «ok».

В результате происходит переход в выбранную папку и отображается ее содержимое, следовательно, функционал является рабочим.

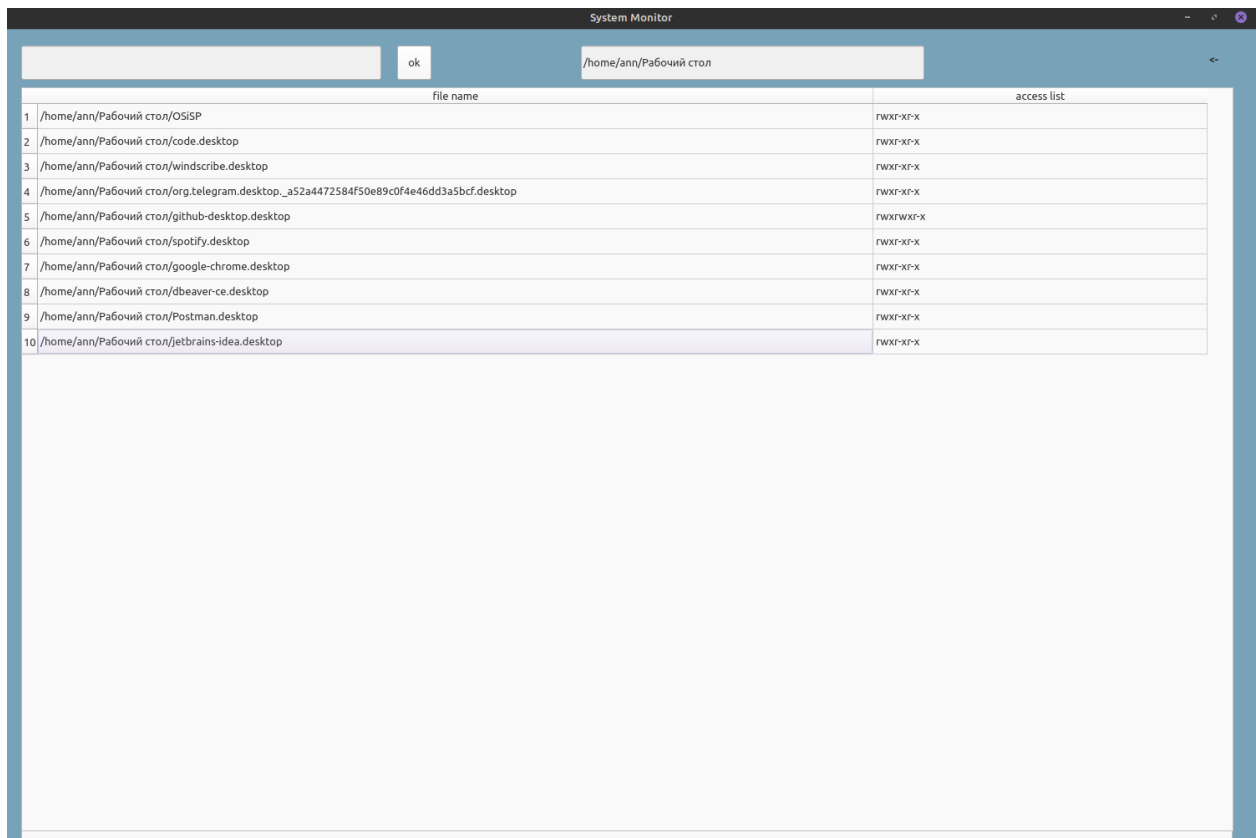


Рисунок 6.2 – Содержимое папки «Рабочий стол»

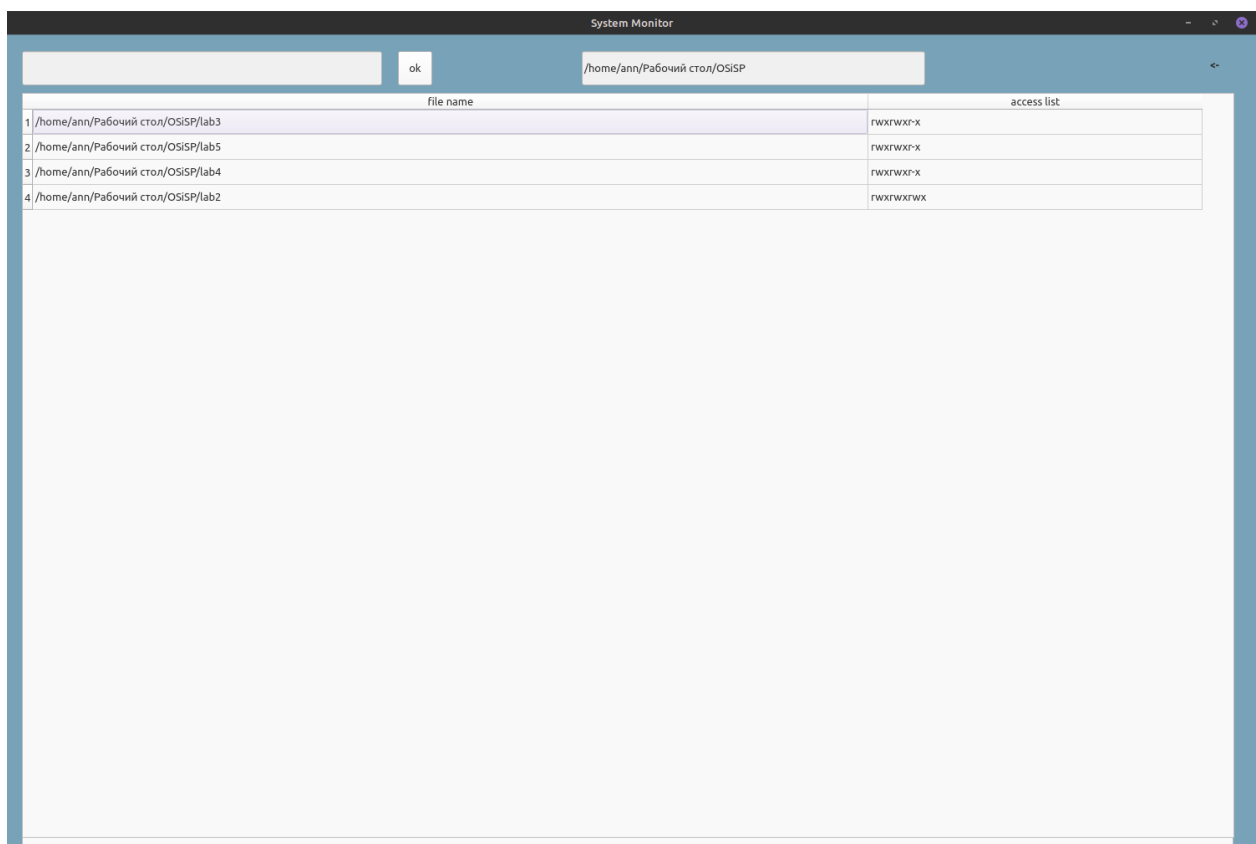
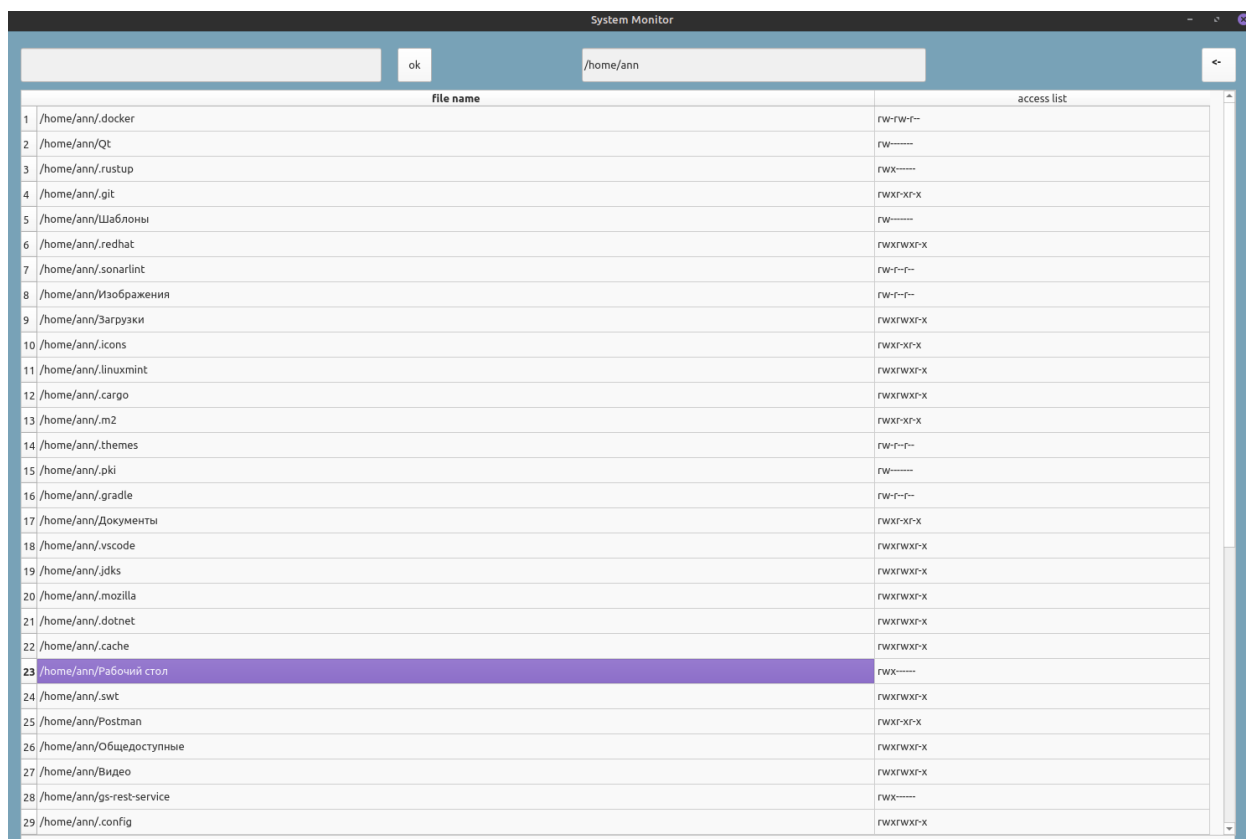


Рисунок 6.3 – Содержимое папки «OSiSP»

Также считаю необходимым протестировать кнопку «<-», которая была неактивной в начальный момент. При наведении мышки подсветка кнопка теперь становится белой, что свидетельствует о том, что она является активной. При нажатии на нее происходит переход на предыдущую директорию, то есть в папку «Рабочий стол». Результат показан на рисунке 6.4.



	file name	access list
1	/home/ann/.docker	rw-rw-r--
2	/home/ann/.Qt	rw-----
3	/home/ann/.rustup	rwX-----
4	/home/ann/.git	rwXr-xr-x
5	/home/ann/Шаблоны	rw-----
6	/home/ann/.redhat	rwXrwXr-x
7	/home/ann/.sonarlint	rw-r--r--
8	/home/ann/Изображения	rw-r--r--
9	/home/ann/Загрузки	rwXrwXr-x
10	/home/ann/.icons	rwXr-xr-x
11	/home/ann/.linuxmint	rwXrwXr-x
12	/home/ann/.cargo	rwXrwXr-x
13	/home/ann/.m2	rwXr-xr-x
14	/home/ann/.themes	rw-r--r--
15	/home/ann/.pki	rw-----
16	/home/ann/.gradle	rw-r--r--
17	/home/ann/Документы	rwXr-xr-x
18	/home/ann/.vscode	rwXrwXr-x
19	/home/ann/.jdk	rwXrwXr-x
20	/home/ann/.mozilla	rwXrwXr-x
21	/home/ann/.dotnet	rwXrwXr-x
22	/home/ann/.cache	rwXrwXr-x
23	/home/ann/Рабочий стол	rwX-----
24	/home/ann/.swt	rwXrwXr-x
25	/home/ann/Postman	rwXr-xr-x
26	/home/ann/Общедоступные	rwXrwXr-x
27	/home/ann/Видео	rwXrwXr-x
28	/home/ann/gs-rest-service	rwX-----
29	/home/ann/.config	rwXrwXr-x

Рисунок 6.4 – Результат работы кнопки «<-»

По результатам проведенного тестирования функционал является полностью рабочим.

ЗАКЛЮЧЕНИЕ

В рамках курсового проекта был разработан системный монитор доступа к файлам для операционной системы Linux. Этот монитор использует список управления доступом для отслеживания и контроля доступа к файлам. В дальнейшем планируется расширение функциональности программы, включая возможность изменения прав доступа к файлам, аналогично команде "chmod" в терминале Linux.

Системный монитор предоставляет пользователю удобный интерфейс для просмотра списка файлов и их прав доступа. Он позволяет пользователям перемещаться по различным папкам и изучать содержимое директорий. Это особенно полезно для администраторов и системных аналитиков, которым необходимо иметь полный контроль над доступом к файлам в системе.

При работе с системным монитором пользователь может видеть список файлов, их имена и соответствующие им права доступа. Это позволяет быстро и удобно отслеживать текущие настройки доступа к файлам. Пользователь может выбирать файлы и папки для более подробного изучения и анализа их прав доступа.

В дополнение к этой функциональности, планируется добавить возможность изменения прав доступа к файлам. Это позволит пользователям монитора изменять права доступа к файлам и папкам непосредственно из пользовательского интерфейса программы. Такая возможность будет полезна при необходимости быстро и удобно изменять права доступа к файлам без необходимости использования командной строки или других инструментов.

Разработанный системный монитор доступа к файлам представляет собой важный инструмент для администраторов и системных аналитиков, работающих с операционной системой Linux. Он обеспечивает удобный способ просмотра и контроля прав доступа к файлам, а также предоставляет возможность изменения этих прав. В будущем, с расширением функциональности программы, она станет еще более мощным инструментом для работы с файловой системой Linux.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Описание системного монитора. – [Электронный ресурс]. – Адрес ресурса: https://en.wikipedia.org/wiki/System_monitor – Дата доступа 05.03.2024
- [2] Таненбаум Э., Остин Т.: Архитектура компьютера. 6-е изд. – СПб.: Питер, 2013. – 816 с.
- [3] Роберт Лав «Linux Системное программирование» 2-е издание, 2014 год
- [4] Объектно-ориентированное программирование в C++, – Роберт Лафоре, 2019
- [5] Qt 5.10 Профессиональное программирование на C++, – Макс Шлее, 2018
- [6] Qt Documentation. – [Электронный ресурс]. -Электронные данные. - Адрес ресурса: <https://doc.qt.io/> - Дата доступа: 20.04.2024

ПРИЛОЖЕНИЕ А
(обязательное)
Схема структурная

ПРИЛОЖЕНИЕ Б
(обязательное)
Диаграмма последовательности

ПРИЛОЖЕНИЕ В
(обязательное)
Схема программы

ПРИЛОЖЕНИЕ Г

(обязательное)

Листинг кода

Файл main.cpp:

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

Файл mainwindow.h:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

public slots:
    void on_pushButton_clicked();

public:
    Ui::MainWindow *ui;
private slots:
    void on_knopka_clicked();
    void on_tableWidget_cellClicked(int row, int column);
};

void outputData(Ui::MainWindow *ui);
#endif // MAINWINDOW_H
```

Файл mainwindow.cpp:

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
```

```

#include "functions.h"
#include <QTableWidget>
#include <QString>
int numberOfDirectories = 0;
int numberOfFiles = 0;
int numberOfPermission = 0;
struct passwd *user = NULL;
string *directoriesArray;
string *filesArray;
string* permissionArray;
string previousDirectory;
string currentDirectory;
int depthDir = 0;
//using namespace std;
MainWindow::MainWindow(QWidget *parent): QMainWindow(parent) , ui(new
Ui::MainWindow)
{
    ui->setupUi(this);
    directoriesArray = new string[1];
    filesArray = new string[1];
    previousDirectory = startPage(previousDirectory, directoriesArray,
filesArray, permissionArray, numberOfDirectories, numberOfFiles,
numberOfPermission);
    currentDirectory = previousDirectory;

    outputData(ui);

}

void outputData(Ui::MainWindow *ui){

    ui->tableWidget->setColumnCount(2);

    ui->tableWidget->setRowCount(numberOfDirectories+numberOfFiles);
    ui->tableWidget->setColumnWidth(0, 1000);
    ui->tableWidget->setColumnWidth(1, 400);

    for(int ridx = 0;ridx< numberOfDirectories+numberOfFiles-1
;ridx++){

        QTableWidgetItem *item = new QTableWidgetItem();
        item->setText(" ");
        ui->tableWidget->setItem(ridx,0, item);

    }
    for(int ridx = 0;ridx< numberOfDirectories+numberOfFiles-1
;ridx++){

        QTableWidgetItem *item = new QTableWidgetItem();
        item->setText(" ");
        ui->tableWidget->setItem(ridx,1, item);

    }
}

```

```

        for(int ridx = 0;ridx< numberOfDirectories ;ridx++){
            QString qstr =
QString::fromStdString(directoriesArray[ridx]);
            QTableWidgetItem *item = new QTableWidgetItem();
            item->setText(qstr);
            ui->tableWidget->setItem(ridx,0, item);

        }
        int counter = 0;
        for (int i = numberOfDirectories; i < numberOfDirectories +
numberOfFiles; i++){
            QString qstr = QString::fromStdString(filesArray[counter]);
            counter++;
            QTableWidgetItem *item = new QTableWidgetItem();
            item->setText(qstr);
            ui->tableWidget->setItem(i,0, item);
        }
        counter =0;
        for(int i=0;i<numberOfDirectories+numberOfFiles;i++){
            QString qstr =
QString::fromStdString(permissionArray[counter]);

            counter++;
            QTableWidgetItem *item = new QTableWidgetItem();
            item->setText(qstr);
            ui->tableWidget->setItem(i, 1, item);
        }
        ui->lineEdit_2->setText(QString::fromStdString(currentDirectory));
        ui->lineEdit->clear();
        if(depthDir ==0){
            ui->knopka->setDisabled(true);
        }
        else {
            ui->knopka->setEnabled(true);
        }
    }

MainWindow::~MainWindow() {
    delete ui;
}

void MainWindow::on_pushButton_clicked()
{

    QString qstr = ui->lineEdit->text();
    string str = qstr.toStdString();
    if (str == ""){
        return;
    }
    depthDir = depthDir+1;
    string searchDirectory = currentDirectory + "/" + str;
    previousDirectory = currentDirectory;
    currentDirectory = searchDirectory;

```

```

        numberOfDirectories = 0;
        numberOfFiles = 0;
        numberOfPermission = 0;
        directoryBrowsing(searchDirectory, directoriesArray, filesArray,
permissionArray, numberOfDirectories, numberOfFiles,
numberOfPermission);
        outputData(ui);
    }

void MainWindow::on_knopka_clicked()
{
    depthDir--;
    numberOfDirectories = 0;
    numberOfFiles = 0;
    numberOfPermission = 0;
    int i = currentDirectory.size();
    char* temp = (char*)calloc(currentDirectory.size()+1,
sizeof(char));
    strcpy(temp, currentDirectory.c_str());

    while(temp[i] != '/') {
        temp[i] = '\0';
        i--;
    }
    temp[i] = '\0';
    currentDirectory = string(temp);

    directoryBrowsing(currentDirectory, directoriesArray, filesArray,
permissionArray, numberOfDirectories, numberOfFiles,
numberOfPermission);
    outputData(ui);
}

void MainWindow::on_tableWidget_cellClicked(int row, int column)
{
    if(column == 1){
        return;
    }
    if (row > numberOfDirectories-1){
        return;
    }
    depthDir++;
    numberOfDirectories = 0;
    numberOfFiles = 0;
    numberOfPermission = 0;
    QTableWidgetItem *qitem = ui->tableWidget->item(row, column);
    QString qstr = qitem->text();
    string str = qstr.toStdString();
    currentDirectory = str;
    directoryBrowsing(str, directoriesArray, filesArray,
permissionArray, numberOfDirectories, numberOfFiles,
numberOfPermission);
    outputData(ui);
}

```

Файл functions.h:

```
#ifndef FUNCTIONS_H
#define FUNCTIONS_H

#endif // FUNCTIONS_H

#include <cstdio>
#include <iostream>
#include <string>

#include <cstdlib>
#include <cerrno>
#include <cstring>
#include <climits>

#include <map>

#include <sys/types.h>
#include <pwd.h>

#include <sys/stat.h>
#include <dirent.h>
#include <unistd.h>

#ifdef _WIN32
#include <io.h>
#elif __linux__
#include <cinttypes>
#include <unistd.h>
#define __int64 int64_t
#define _close close
#define _read read
#define _lseek64 lseek64
#define _O_RDONLY O_RDONLY
#define _open open
#define _lseeki64 lseek64
#define _lseek lseek
#define stricmp strcasecmp
#endif

using namespace std;

string startPage(string &previousDirectory, string* &directoriesArray,
string* &filesArray, string* &permissionArray,
                int &numberOfDirectories, int &numberOfFiles, int
&numberOfPermission);

char* displayPermission (int st_mode );

unsigned directoryBrowsing(string introducedDir, string*
&directoriesArray, string* &filesArray, string* &permissionArray,
                int &numberOfDirectories, int
&numberOfFiles, int &numberOfPermission);
```


Файл functions.cpp:

```
#include "functions.h"

string startPage(string &previousDirectory, string* &directoriesArray,
string* &filesArray, string* &permissionArray,
                int &numberOfDirectories, int &numberOfFiles, int
&numberOfPermission){

    directoriesArray = new string[1];
    filesArray = new string[1];
    permissionArray = new string[1];

    struct passwd *user = NULL;

    string temp;

    uid_t user_id = getuid();
    user = getpwuid(user_id);

    string direct;

    direct = user->pw_dir;
    previousDirectory = direct;

    directoryBrowsing(direct, directoriesArray, filesArray,
permissionArray, numberOfDirectories, numberOfFiles,
numberOfPermission);
    return previousDirectory;

}

char* displayPermission (int st_mode )
{
    static const char xtbl[10] = "rwxrwxrwx";
    char*         amode = (char*)calloc(10, sizeof(char));
    int           i, j;

    for ( i = 0, j = ( 1 << 8 ); i < 9; i++, j >>= 1 )
        amode[i] = ( st_mode&j ) ? xtbl[i]: '-';
    if ( st_mode & S_ISUID )    amode[2]= 's';
    if ( st_mode & S_ISGID )    amode[5]= 's';
    if ( st_mode & S_ISVTX )    amode[8]= 't';
    amode[9]='\0';
    return amode;

}

unsigned directoryBrowsing(string introducedDir, string*
&directoriesArray, string* &filesArray, string* &permissionArray,
                int &numberOfDirectories, int
&numberOfFiles, int &numberOfPermission){
    int accessParameters;
    DIR *dir = NULL;
    string* temp;
```

```

string resultPermission;
struct dirent *entry = NULL;
string pathName;
mode_t status;
dir = opendir(introducedDir.c_str());
if( dir == NULL ) {
    return 0;
}
entry = readdir(dir);
while(entry != NULL) {
    struct stat entryInfo;
    if((strcmp(entry->d_name, ".") == 0) ||
(strcmp(entry->d_name, "..") == 0)) {
        entry = readdir(dir);
        continue;
    }
    pathName = introducedDir;
    pathName += "/";
    pathName += entry->d_name;
    if(lstat(pathName.c_str(), &entryInfo) == 0) {
        if(S_ISDIR(entryInfo.st_mode)) {
            directoriesArray[numberOfDirectories] = pathName;
            numberOfDirectories++;
            temp = new string[numberOfDirectories + 1];
            for (int i = 0; i < numberOfDirectories; i++) {
                temp[i] = directoriesArray[i];
            }
            directoriesArray = temp;

            resultPermission =
string(displayPermission(entryInfo.st_mode));
            permissionArray[numberOfPermission] =
resultPermission;
            numberOfPermission++;
            temp = new string[numberOfPermission + 1];
            for (int i = 0; i < numberOfPermission; i++){
                temp[i] = permissionArray[i];
            }
            permissionArray = temp;
        }
        else if(S_ISREG(entryInfo.st_mode)) {
            filesArray[numberOfFiles] = pathName;
            numberOfFiles++;
            temp = new string[numberOfFiles + 1];
            for (int i = 0; i < numberOfFiles; i++) {
                temp[i] = filesArray[i];
            }
            filesArray = temp;

            resultPermission =
string(displayPermission(entryInfo.st_mode));
            permissionArray[numberOfPermission] =
resultPermission;
            numberOfPermission++;
            temp = new string[numberOfPermission + 1];
            for (int i = 0; i < numberOfPermission; i++){

```

```

        temp[i] = permissionArray[i];
    }
    permissionArray = temp;

}
else if(S_ISLNK(entryInfo.st_mode)) {
    //string targetName;
    char targetName[PATH_MAX + 1];
    if(readlink(pathName.c_str(), targetName, PATH_MAX) !=
-1) {

        filesArray[numberOfFiles] = pathName;
        numberOfFiles++;
        temp = new string[numberOfFiles + 1];
        for (int i = 0; i < numberOfFiles; i++) {
            temp[i] = filesArray[i];
        }
        filesArray = temp;
    }
    else {
        printf("\t%s -> (invalid symbolic link!)\n",
pathName);
    }
}
}
else {
    printf("Error statting %s: %s\n", pathName,
strerror(errno));
}
    entry = readdir(dir);
}
    (void)closedir(dir);
    return 0;
}

```

ПРИЛОЖЕНИЕ Д
(обязательное)
Ведомость документов