**СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ** 12](#_Toc198805052)

[**1 ОБЗОР ИСТОЧНИКОВ ЛИТЕРАТУРЫ** 14](#_Toc198805053)

[**1.1 Анализ существующих аналогов** 14](#_Toc198805054)

[**1.2** **Постановка задачи** 16](#_Toc198805055)

[**1.3** **Обзор методов и алгоритмов решения поставленной задачи** 17](#_Toc198805056)

[**2** **СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ** 19](#_Toc198805057)

[**2.1Мониторинг изменений файлов** 19](#_Toc198805058)

[**2.2 Проверка целостности файлов** 19](#_Toc198805059)

[**2.3 Многопоточность и параллелизм** 20](#_Toc198805060)

[**2.4 Фоновый режим и масштабируемость** 20](#_Toc198805061)

[**2.5 Обработка ошибок и восстановление** 21](#_Toc198805062)

[**3** **ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ** 22](#_Toc198805063)

[**3.1** **Алгоритм удаления файлов** 22](#_Toc198805064)

[**3.2** **Алгоритм добавления файлов** 23](#_Toc198805065)

[**4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ** 24](#_Toc198805082)

[**4.1 Класс FileMonitor** 24](#_Toc198805083)

[**4.2** **Функции модуля Monitoring** 24](#_Toc198805100)

[**4.3 Класс UI** 25](#_Toc198805108)

[**4.4 Функции модуля Navigation** 25](#_Toc198805114)

[**4.5 Функции модуля Utils** 25](#_Toc198805121)

[**4.6 Основная программа (main.cpp)** 26](#_Toc198805126)

[**4.7 Перечисление ColorPairs** 26](#_Toc198805132)

[**4.8 Структуры данных** 26](#_Toc198805140)

[**4.9 Потоки и синхронизация** 26](#_Toc198805143)

[**4.10 Обработка ошибок** 27](#_Toc198805146)

[**5** **ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ** 28](#_Toc198805150)

[**6** **РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ** 29](#_Toc198805151)

[**6.1** **Добавление файла в отслеживание** 29](#_Toc198805157)

[**6.2** **Удаление файла из отслеживания** 30](#_Toc198805160)

[**6.3** **Начало и остановка мониторинга** 30](#_Toc198805162)

[**6.4** **Просмотр текущих отслеживаемых файлов** 30](#_Toc198805165)

[**6.5** **Возможные ошибки и уведомления** 31](#_Toc198805168)

[**ЗАКЛЮЧЕНИЕ** 32](#_Toc198805176)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** 34](#_Toc198805177)

[**ПРИЛОЖЕНИЕ А** 35](#_Toc198805188)

[**ПРИЛОЖЕНИЕ Б** 36](#_Toc198805191)

[**ПРИЛОЖЕНИЕ В** 37](#_Toc198805194)

[**ПРИЛОЖЕНИЕ Г** 38](#_Toc198805197)

[**ПРИЛОЖЕНИЕ Д** 39](#_Toc198805200)

# **ВВЕДЕНИЕ**

В условиях стремительного увеличения объемов информации и ужесточения требований к безопасности данных разработка программных инструментов для контроля изменений и обеспечения целостности файлов становится все более актуальной. Защита данных от несанкционированных изменений и поддержание их целостности являются одними из ключевых задач современных информационных систем. Важность этой задачи обусловлена необходимостью надежного хранения критически важной информации, предотвращения ее утери и минимизации рисков, связанных с несанкционированным доступом.

Этот проект направлен на создание многопоточной программы для фонового мониторинга изменений и проверки целостности группы файлов. Основная цель программы – обеспечить непрерывное наблюдение за состоянием файловой системы и оперативное выявление любых изменений. Применение многопоточности позволяет значительно повысить производительность и эффективность программы, обеспечивая параллельное выполнение задач и сокращение времени отклика.

Многопоточность – это подход к разработке программ, при котором выполнение задач разделяется на несколько потоков, работающих параллельно. Это позволяет эффективно использовать ресурсы процессора, особенно в системах с несколькими ядрами. В рамках данной программы многопоточность будет реализована с использованием стандартных библиотек C++, таких как std::thread, std::mutex и std::unique\_lock. Эти инструменты позволяют создавать и управлять потоками, а также синхронизировать доступ к общим ресурсам, предотвращая конфликты. Дополнительно будут использоваться семафоры, барьеры и другие синхронизирующие примитивы для более гибкого управления потоками.

Фоновые программы – это приложения, которые работают без активного участия пользователя, выполняя задачи в фоновом режиме. Они часто используются для мониторинга, обслуживания системы или выполнения длительных операций. В данной работе программа будет работать в фоновом режиме, непрерывно отслеживая изменения в файловой системе и проверяя целостность файлов.

Для контроля целостности файлов будут использоваться алгоритмы хеширования, такие как MD5, SHA-1 и SHA-256. Эти алгоритмы позволяют создавать уникальные хеш-суммы для каждого файла, которые затем сравниваются с ранее сохраненными значениями. В случае обнаружения изменений программа сможет оперативно уведомить пользователя или систему о потенциальных угрозах.

Для мониторинга изменений в файловой системе будут использоваться платформенно-зависимые инструменты, такие как inotify для Linux и FileSystemWatcher для Windows. Эти средства позволяют отслеживать события изменения файлов и каталогов в реальном времени, обеспечивая оперативное реагирование на любые изменения.

Данная курсовая работа направлена на развитие методов обеспечения информационной безопасности и предлагает эффективное решение для защиты данных в условиях динамично изменяющейся информационной среды. Реализация многопоточной программы для контроля изменений и целостности файлов позволит не только повысить уровень безопасности данных, но и обеспечить их надежное хранение и оперативное выявление любых угроз.

# **1 ОБЗОР ИСТОЧНИКОВ ЛИТЕРАТУРЫ**

## **1.1 Анализ существующих аналогов**

Разработка программного обеспечения для контроля изменений и целостности файлов является важной задачей в области информационной безопасности. На рынке существует множество решений, которые предлагают различные подходы к реализации данной функциональности. Ниже приведен анализ существующих аналогов, их преимуществ и недостатков.

* + 1. Tripwire – одна из самых известных программ для контроля целостности файлов. Она используется для мониторинга изменений в файловой системе и сравнения текущего состояния файлов с эталонными значениями, хранящимися в базе данных.

Особенности: использует алгоритмы хеширования (MD5, SHA) для создания контрольных сумм файлов. Поддерживает многоплатформенность (Linux, Windows), позволяет настраивать политики мониторинга для отдельных файлов или каталогов. Преимущества: высокая надежность и точность обнаружения изменений. Широкие возможности настройки и интеграции с другими системами. Поддержка многопоточности для повышения производительности. Недостатки: сложность настройки и управления для новичков. Высокая стоимость коммерческой версии. Отсутствие встроенной поддержки фонового мониторинга в реальном времени (требует периодического запуска).

* + 1. AIDE – это open-source инструмент для контроля целостности файлов, который часто используется в Unix-системах. Он создает базу данных контрольных сумм файлов и сравнивает ее с текущим состоянием системы.

Особенности: поддерживает различные алгоритмы хеширования (SHA-1, SHA-256, MD5). Позволяет настраивать правила для мониторинга конкретных файлов и каталогов. Интегрируется с системами логирования для уведомлений об изменениях. Преимущества: бесплатное и открытое программное обеспечение. Легкость в установке и настройке. Поддержка многопоточности для ускорения работы. Недостатки: отсутствие встроенной поддержки фонового мониторинга в реальном времени. Ограниченная функциональность по сравнению с коммерческими аналогами. Нет графического интерфейса, что усложняет использование для неопытных пользователей.

* + 1. OSSEC – это система обнаружения вторжений с открытым исходным кодом, которая включает в себя функциональность контроля целостности файлов. Она позволяет отслеживать изменения в файловой системе и уведомлять администратора о подозрительных действиях.

Особенности: поддержка многопоточности для обработки событий в реальном времени. Интеграция с системами логирования и уведомлений (email, SMS). Возможность настройки правил для мониторинга конкретных файлов и каталогов. Преимущества: бесплатное и открытое программное обеспечение. Широкие возможности настройки и интеграции. Поддержка многоплатформенности (Linux, Windows, macOS). Недостатки: сложность настройки и управления. Ограниченная производительность при работе с большими объемами данных. Отсутствие встроенной поддержки многопоточности для фонового мониторинга.

* + 1. FileAudit – это коммерческое решение для мониторинга и аудита изменений в файловой системе. Оно предоставляет подробные отчеты о всех изменениях файлов и каталогов.

Особенности: поддержка многопоточности для обработки событий в реальном времени. Интеграция с системами уведомлений (email, SMS). Подробные отчеты и графический интерфейс для анализа данных. Преимущества: высокая производительность и надежность. Удобный графический интерфейс для настройки и управления. Поддержка многоплатформенности (Windows, Linux). Недостатки: высокая стоимость лицензии. Ограниченная гибкость в настройке правил мониторинга. Отсутствие поддержки open-source сообщества.

* + 1. Inotify (Linux) и FileSystemWatcher (Windows) – это встроенные инструменты для мониторинга изменений в файловой системе в реальном времени. Они используются для создания собственных решений контроля целостности файлов.

Особенности: поддержка многопоточности для обработки событий в реальном времени. Легкость интеграции с другими системами. Возможность настройки правил для мониторинга конкретных файлов и каталогов. Преимущества: бесплатное и встроенное решение. Высокая производительность и низкие накладные расходы. Поддержка многопоточности для обработки событий. Недостатки: требует разработки собственного решения для контроля целостности файлов. Отсутствие встроенной поддержки алгоритмов хеширования. Ограниченная функциональность по сравнению с готовыми решениями.

Существующие аналоги предлагают различные подходы к решению задачи контроля изменений и целостности файлов. Коммерческие решения, такие как Tripwire и FileAudit, предоставляют высокую надежность и удобство использования, но имеют высокую стоимость. Open-source инструменты, такие как AIDE и OSSEC, предлагают гибкость и бесплатное использование, но требуют значительных усилий для настройки и интеграции. Встроенные инструменты, такие как Inotify и FileSystemWatcher, позволяют создавать собственные решения, но требуют разработки дополнительной функциональности.

Разрабатываемая программа должна объединить преимущества существующих решений: многопоточность, фоновый мониторинг в реальном времени, поддержку алгоритмов хеширования и простоту настройки. Это позволит создать конкурентоспособное решение для контроля целостности файлов в современных информационных системах.

* 1. **Постановка задачи**

В рамках проекта будет осуществлено проектирование и разработка многопоточной программы для фонового контроля изменений и целостности группы файлов. Основной задачей является обеспечение непрерывного мониторинга состояния файловой системы с использованием многопоточности, что позволит программе эффективно отслеживать изменения в файлах и каталогах в режиме реального времени. Для реализации многопоточности будут использоваться POSIX Threads (pthread), что обеспечит параллельную обработку задач и повысит производительность программы.

Одной из ключевых функций программы станет контроль целостности файлов. Для этого будет реализовано хеширование файлов с использованием алгоритмов, таких как MD5, SHA-1 и SHA-256. Программа будет сравнивать текущие хеш-суммы файлов с ранее сохраненными значениями, что позволит оперативно выявлять любые изменения и обеспечивать целостность данных.

Для повышения удобства использования программа будет оснащена механизмами уведомлений о выявленных изменениях, включая отправку оповещений по электронной почте или через системные уведомления. Это обеспечит своевременное информирование пользователя или администратора о потенциальных угрозах или несанкционированных изменениях.

В ходе разработки особое внимание будет уделено оптимизации производительности программы и обеспечению её стабильной работы в условиях высокой нагрузки. Программа будет интегрирована с ядром операционной системы для эффективного управления системными ресурсами, что позволит минимизировать накладные расходы и обеспечить надёжную работу в многозадачной среде.

* 1. **Обзор методов и алгоритмов решения поставленной задачи**

Реализация многопоточной программы для фонового мониторинга изменений и проверки целостности файлов требует тщательного выбора методов и алгоритмов, которые обеспечат не только высокую производительность, но и надежность работы системы. В основе проектирования лежит стремление объединить современные технологии с проверенными подходами, что позволяет создать гибкое решение, адаптированное под различные сценарии использования.

* + 1. Мониторинг изменений файлов. Ключевым элементом системы является механизм отслеживания изменений в файловой системе. Для этой цели используются платформенно-зависимые API, такие как inotify в Linux и FileSystemWatcher в Windows, которые предоставляют возможность реагировать на события в реальном времени — будь то создание, изменение или удаление файлов. Эти инструменты интегрированы непосредственно в ядро операционных систем, что минимизирует задержки и обеспечивает высокую эффективность. Однако, учитывая возможные сбои в работе API, разработчики предусмотрели резервный механизм — периодическое сканирование файловой системы. Этот подход, основанный на рекурсивном обходе каталогов с заданным интервалом, гарантирует, что даже в случае временной недоступности системных средств мониторинга, изменения не останутся незамеченными. Таким образом, гибридная стратегия сочетает преимущества оперативного реагирования и надежности.
    2. Проверка целостности данных. Для верификации целостности файлов система опирается на криптографические хеш-функции, которые преобразуют содержимое файлов в уникальные цифровые отпечатки. Основное предпочтение отдается алгоритму SHA-256, признанному отраслевым стандартом благодаря своей устойчивости к коллизиям и высокому уровню безопасности. В качестве альтернативы, для ускорения процессов обработки крупных файлов, используется BLAKE3 — современный алгоритм, сочетающий скорость вычислений с криптографической надежностью. Для обеспечения совместимости с устаревшими системами предусмотрена поддержка MD5, несмотря на его известные уязвимости. Дополнительно, чтобы снизить нагрузку на ресурсы при работе с большими объемами данных, реализована инкрементальная проверка. Этот метод позволяет пересчитывать хеш-суммы только для измененных блоков файлов, что существенно экономит время и вычислительные мощности. В сценариях, где скорость критически важна, применяется легковесный алгоритм CRC32, обеспечивающий базовый уровень проверки без избыточных затрат.
    3. Многопоточность и управление ресурсами. Эффективное использование многопоточности — один из краеугольных камней данной системы. Для параллельной обработки задач задействован пул потоков (Thread Pool), который распределяет задания между рабочими потоками, минимизируя простои и повышая общую производительность. Синхронизация доступа к общим ресурсам, таким как очередь событий или база данных хеш-сумм, осуществляется через мьютексы и семафоры, что предотвращает возникновение конфликтов и race condition. Отдельное внимание уделено асинхронной обработке: мониторинг изменений и проверка целостности выполняются в независимых потоках, что позволяет системе оставаться отзывчивой даже при высокой нагрузке. Такая архитектура не только ускоряет выполнение задач, но и обеспечивает стабильность работы в многозадачной среде.
    4. Обработка ошибок и восстановление. Надежность системы подкреплена комплексным подходом к обработке сбоев. Все критические операции, такие как обновление хеш-сумм или запись данных в лог, выполняются транзакционно, что гарантирует целостность данных даже в случае непредвиденных ошибок. Для быстрого восстановления после сбоев реализованы чекпоинты — периодические сохранения состояния системы, позволяющие продолжить работу с последней устойчивой точки. Детальное логирование событий с разделением на уровни важности (INFO, WARNING, ERROR) обеспечивает прозрачность работы и упрощает диагностику проблем. В случае обнаружения нарушений целостности система способна автоматически реагировать — например, восстанавливать файлы из резервных копий или инициировать повторную проверку. Эти меры направлены на минимизацию вмешательства пользователя и поддержание непрерывности работы.

Выбранные методы и алгоритмы отражают баланс между инновациями и практичностью. Использование платформенно-ориентированных API обеспечивает глубокую интеграцию с операционными системами, а гибкий подход к хешированию позволяет адаптировать программу под разнообразные требования — от критически важных систем с повышенными стандартами безопасности до сценариев, где приоритетом является скорость обработки. Внедрение многопоточности и продуманные механизмы обработки ошибок делают решение не только производительным, но и устойчивым к внешним воздействиям. Все это в совокупности формирует надежный инструмент для защиты данных в условиях динамично меняющейся цифровой среды.

1. **СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ**

Рассмотрим архитектуру приложения многопоточной программы для фонового контроля изменений целостности группы файлов.

**2.1Мониторинг изменений файлов**

Основу системы составляет механизм отслеживания изменений (мониторинг изменений файлов – механизм отслеживания изменений в файловой системе (создание, удаление, модификация, перемещение файлов) в реальном времени или с заданной периодичностью), который может работать как в реальном времени, так и в режиме периодического сканирования.

В Unix-подобных системах для этого используется inotify – эффективный API ядра Linux, позволяющий приложениям получать уведомления о любых событиях файловой системы. В Windows аналогичную функциональность предоставляет ReadDirectoryChangesW, а в macOS – FSEvents.

Система предусматривает резервный механизм мониторинга, основанный на периодическом рекурсивном обходе дерева каталогов и сравнении текущего состояния с предыдущим. Это обеспечивает дополнительную надёжность в случаях, когда системные вызовы недоступны или не покрывают требуемый набор событий.

Обнаруженные события помещаются в очередь (Event Queue), что позволяет изолировать процессы обнаружения изменений от их обработки, обеспечивая масштабируемость и устойчивость к нагрузке.

**2.2 Проверка целостности файлов**

Проверка целостности файлов – процесс верификации файлов на предмет нежелательных изменений с использованием криптографических хеш-функций или цифровых подписей.

Для механизма верификации целостности данных применяются криптографические хеш-функции (SHA-256, BLAKE3), которые преобразуют содержимое файла в уникальную строку фиксированной длины. Полученный хеш сравнивается с эталонным значением, хранящимся в защищенном хранилище.

Для ускорения операций и снижения нагрузки на систему реализована поддержка инкрементальной проверки: при обнаружении изменений хеш пересчитывается только для изменённых блоков файла, что особенно актуально для работы с большими файлами. Это реализуется с помощью блоковой адресации и использования буферизованного чтения или отображения файлов в память (mmap).

Для задач с пониженными требованиями к безопасности могут применяться легковесные алгоритмы, такие как CRC32. Также возможно использование цифровых подписей для проверки подлинности и неизменности файла в средах с повышенными требованиями к защите данных.

**2.3 Многопоточность и параллелизм**

Многопоточность и параллелизм – способ организации выполнения задач в нескольких потоках для ускорения обработки данных. Основной поток отвечает за координацию работы, в то время как пул рабочих потоков (Thread Pool) занимается обработкой задач из очереди событий. Для предотвращения гонки данных (Race Condition) при доступе к общим ресурсам применяются мьютексы (Mutex) и другие механизмы синхронизации. Особое внимание уделяется минимизации блокировок, чтобы избежать снижения производительности в условиях высокой нагрузки.

**2.4 Фоновый режим и масштабируемость**

Программа функционирует в фоновом режиме, что позволяет ей работать без вмешательства пользователя и минимально нагружать ресурсы системы. В Unix-подобных ОС система запускается как демон, а в Windows — как служба (Windows Service). Это обеспечивает непрерывный контроль за файлами и возможность автоматического запуска при старте системы.

Для повышения масштабируемости при работе с большим количеством файлов реализовано горизонтальное разделение задач — шардирование (sharding). Файлы делятся на группы, которые обрабатываются разными потоками либо, в случае распределённых систем, — отдельными узлами (workers). В таком случае применяется архитектурный шаблон Master-Worker: главный узел (Master) управляет координацией и балансировкой нагрузки, распределяя задачи между подчинёнными агентами.

Также предусмотрена поддержка динамической конфигурации, позволяющей добавлять или удалять каталоги без перезапуска системы.

**2.5 Обработка ошибок и восстановление**

Надёжность системы обеспечивается за счёт устойчивости к ошибкам и поддержке механизмов восстановления. Все критически важные операции (например, добавление/удаление отслеживаемых файлов, обновление хешей) выполняются атомарно и сопровождаются логированием. В случае непредвиденных сбоев (например, при потере доступа к хранилищу или сбоях файловой системы) реализована возможность автоматического восстановления на основе контрольных точек (checkpoints).

Логирование ведётся в отдельные файлы (file\_monitor.log, changes.log) с возможностью ротации и архивирования. Лог содержит подробную информацию о типе события, времени возникновения, объекте и результате предпринятых действий. Это существенно облегчает отладку, аудит и расследование инцидентов.

В случае обнаружения нарушения целостности система может автоматически выполнять предопределённые действия: отправка уведомлений (email, Telegram), блокировка доступа, восстановление из резервной копии. Для этого предусмотрена интеграция с подсистемами резервного копирования и оповещения администратора.

1. **ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ**
   1. **Алгоритм удаления файлов**

Схема алгоритма приведена в чертеже ГУИР.6-05-0611-05.316 ПД1.

Описание алгоритма по шагам:

**Шаг** **1** Начало выполнения функции removeFileFromWatch.

**Шаг** **2** Вывод отладочной информации: имя файла и параметры isMonitoring и inotifyFd.

**Шаг** **3** Поиск файла в множестве отслеживаемых файлов (trackedFiles).

**Шаг** **4** Если файл не найден, вывод сообщения об ошибке, завершение функции (переход к шагу 18).

**Шаг** **5** Инициализация переменной wdToRemove значением -1 (идентификатор дескриптора наблюдения для удаления).

**Шаг** **6** Поиск в отображении watchDescriptors пары, соответствующей данному пути к файлу (filePath).

**Шаг** **7** Если дескриптор найден, сохранение его в wdToRemove.

**Шаг** **8** Проверка: найден ли дескриптор наблюдения (значение wdToRemove не равно -1).

**Шаг** **9** Если дескриптор найден, вывод отладочной информации.

**Шаг** **10** Проверка условий: включён ли режим мониторинга (isMonitoring == true) и корректен ли файловый дескриптор (inotifyFd != -1).

**Шаг** **11** Если оба условия выполнены, вызов системной функции inotify\_rm\_watch для удаления наблюдения.

**Шаг** **12** Проверка результата вызова inotify\_rm\_watch. Если возврат -1, вывод сообщения об ошибке с помощью perror. Иначе вывод сообщения об успешном удалении дескриптора.

**Шаг** **13** Если условия шага 10 не выполнены, вывод сообщения о том, что удаление наблюдения пропущено из-за неактивного мониторинга или некорректного дескриптора.

**Шаг** **14** Удаление записи о дескрипторе из отображения watchDescriptors.

**Шаг 15** Вывод отладочной информации о завершении удаления дескриптора.

**Шаг** **16** Если дескриптор не был найден (значение wdToRemove == -1), вывод сообщения об ошибке.

**Шаг** **17** Удаление файла из множества trackedFiles.

**Шаг** **18** Вывод отладочного сообщения о завершении функции removeFileFromWatch.

**Шаг** **19** Конец.

* 1. **Алгоритм добавления файлов**

Схема алгоритма приведена в чертеже ГУИР.6-05-0611-05.316 ПД2.

Описание алгоритма по шагам:

Шаг 1 Начало выполнения функции addFileToWatch.

Шаг 2 Получение эксклюзивного доступа к разделяемым структурам данных с использованием механизма lock\_guard и мьютекса mtx.

Шаг 3 Поиск пути filePath в множестве trackedFiles (уже отслеживаемые файлы).

Шаг 4 Если файл уже отслеживается, вывод сообщения и завершение функции (переход к шагу 10).

Шаг 5 Вызов системной функции inotify\_add\_watch, которая добавляет наблюдение за файлом filePath. Указываются маски событий:

– IN\_MODIFY – изменение содержимого файла,

– IN\_CLOSE\_WRITE – закрытие файла после записи,

– IN\_MOVED\_TO – перемещение файла в наблюдаемую директорию.

Шаг 6 Проверка результата вызова inotify\_add\_watch. Если возвращённое значение wd равно -1, это означает ошибку.

Шаг 7 В случае ошибки добавления вывода сообщения об ошибке и завершение функции (переход к шагу 10).

Шаг 8 Добавление пути filePath в множество trackedFiles.

Шаг 9 Добавление пары дескриптор-наблюдения → путь файла в отображение watchDescriptors.

Шаг 10 Вывод сообщения о том, что файл успешно добавлен к отслеживанию.

Шаг 11 Конец.

1. **РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ**

**4.1 Класс FileMonitor**

Класс FileMonitor отвечает за мониторинг изменений файлов с использованием API inotify (Linux) и управления списком отслеживаемых файлов. Определён в файлах FileMonitor.h и FileMonitor.cpp.

Поля класса:

1. int inotifyFd – дескриптор inotify.
2. bool isMonitoring – флаг активности мониторинга.
3. unordered\_set<string> trackedFiles – список отслеживаемых файлов.
4. unordered\_map<int, string> watchDescriptors – таблица дескрипторов inotify.
5. mutex mtx – мьютекс для синхронизации потоков.

Методы класса:

1. FileMonitor() – конструктор: инициализирует inotify, создаёт директорию для бэкапов, загружает список отслеживаемых файлов.
2. ~FileMonitor() – деструктор: останавливает мониторинг, сохраняет список файлов, закрывает дескрипторы.
3. addFile(const string& filePath) – добавляет файл в список отслеживания.
4. removeFile(const string& filePath) – удаляет файл из списка.
5. startMonitoring() – запускает поток мониторинга.
6. stopMonitoring() – останавливает поток.
7. listTrackedFiles() – выводит список отслеживаемых файлов.
8. browseAndSelectFile(const string& startDir) – открывает интерфейс выбора файла.
   1. **Функции модуля Monitoring**

Функции модуля Monitoring реализуют низкоуровневую логику работы с inotify, управление потоками и обработку событий. Определены в файлах Monitoring.h и Monitoring.cpp.

Функции:

addFileToWatch() – добавляет файл в отслеживание через inotify.

removeFileFromWatch() – удаляет файл из отслеживания.

startMonitoringThread() – запускает поток для обработки событий inotify.

stopMonitoringThread() – останавливает поток.

listTrackedFilesImpl() – выводит список файлов с синхронизацией через мьютекс.

**4.3 Класс UI**

Класс UI реализует текстовый интерфейс для навигации по файловой системе и выбора файлов с использованием библиотеки ncurses. Определён в файлах UI.h и UI.cpp.

Функции:

initColors() – инициализирует цветовые схемы.

printColored() – выводит текст с цветом и подсветкой.

browseAndSelectFileImpl() – отображает интерактивный интерфейс выбора файлов.

**4.4 Функции модуля Navigation**

Модуль Navigation предоставляет функционал для работы с историей навигации. Определён в файлах Navigation.h и Navigation.cpp.

Функции:

showHistory() – отображает историю посещённых директорий в отдельном окне ncurses.

Параметры:

deque<string>& dirHistory – очередь для хранения истории.

MAX\_HISTORY\_ITEMS – ограничение на количество записей (50).

**4.5 Функции модуля Utils**

Модуль Utils содержит вспомогательные функции для работы с терминалом и файлами. Определён в файлах Utils.h и Utils.cpp.

Функции:

clearScreen() – очищает терминал.

getFileType(char type) – возвращает тип файла в виде строки.

**4.6 Основная программа (main.cpp)**

Файл main.cpp содержит точку входа в программу и управляет режимами работы (интерактивный/фоновый).

Функции:

isAnotherInstanceRunning() – проверяет, запущен ли уже процесс мониторинга.

createLockFile() – создаёт файл блокировки с PID процесса.

main() – обрабатывает аргументы командной строки, запускает мониторинг, предоставляет меню выбора действий.

**4.7 Перечисление ColorPairs**

Перечисление ColorPairs задаёт цветовые схемы для текстового интерфейса. Используется библиотекой ncurses для оформления элементов. Определено в UI.h.

Значения:

COLOR\_DIR – зелёный для директорий.

COLOR\_FILE – белый для файлов.

COLOR\_LINK – голубой для ссылок.

COLOR\_HEADER – жёлтый для заголовков.

COLOR\_HIGHLIGHT – чёрный на белом фоне для выделения.

**4.8 Структуры данных**

deque<string> dirHistory – хранит историю навигации по директориям.

stack<string> backStack – стек для возврата к предыдущим директориям.

**4.9 Потоки и синхронизация**

std::thread monitoringThread – поток для обработки событий inotify.

std::mutex mtx – синхронизирует доступ к общим ресурсам (списки файлов, дескрипторы).

**4.10 Обработка ошибок**

Транзакционность операций – гарантирует целостность данных при записи в лог-файлы.

Логирование – запись ошибок в file\_monitor.err с указанием времени и причины.

Восстановление – автоматическое создание резервных копий и чекпоинтов.

1. **ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ**
2. **РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

Это руководство предназначено для пользователей консольной программы отслеживания изменений в файлах – утилиты, реализующей мониторинг определённых файлов с использованием системного интерфейса inotify. Здесь описаны шаги по установке, запуску и использованию программы, а также основные доступные действия.

Для работы достаточно иметь скомпилированную программу, расположенную в директории без кириллических символов, на компьютере с ОС на базе Unix (например, Linux) (рис. 6.1).

Рисунок 6.1 – Запуск программы в терминале

Программа использует текстовый интерфейс командной строки (TUI). Все действия выполняются через ввод текстовых команд и путей к файлам. После запуска пользователю предоставляется выбор доступных операций: добавить файл в отслеживание, удалить файл из отслеживания, просмотреть список отслеживаемых файлов, начать или остановить мониторинг (рис. 6.2).

Рисунок 6.2 – Главное меню программы

* 1. **Добавление файла в отслеживание**

Для начала мониторинга изменений конкретного файла необходимо выбрать опцию добавления файла. Пользователь вводит абсолютный или относительный путь к отслеживаемому файлу. Программа проверяет, не отслеживается ли файл уже. Если нет — устанавливается inotify-наблюдение на события изменения (IN\_MODIFY), закрытия после записи (IN\_CLOSE\_WRITE) и перемещения файла в директорию (IN\_MOVED\_TO).

После успешного добавления информация сохраняется во внутренние структуры программы: множество отслеживаемых файлов (trackedFiles) и отображение дескрипторов (watchDescriptors). Программа сообщает о успешном добавлении с указанием номера watch descriptor.

* 1. **Удаление файла из отслеживания**

Удаление файла из мониторинга производится через соответствующую опцию меню. Пользователь указывает путь к нужному файлу. Программа проверяет наличие файла в списке отслеживаемых. Если файл найден, и мониторинг активен — выполняется вызов inotify\_rm\_watch с дескриптором, соответствующим данному пути. В случае успеха дескриптор удаляется из хранилища, а путь — из множества отслеживаемых файлов. В противном случае пользователю выводится сообщение об ошибке (например, если файл не отслеживался ранее).

* 1. **Начало и остановка мониторинга**

Для запуска постоянного наблюдения за файлами необходимо активировать режим мониторинга. В этом режиме программа регистрирует события, происходящие с файлами, на которые ранее были установлены наблюдения. Все события отображаются в реальном времени, с указанием времени, типа события и пути к файлу.

Остановка мониторинга доступна через соответствующую команду. После завершения наблюдения пользователь может внести изменения в список отслеживаемых файлов.

* 1. **Просмотр текущих отслеживаемых файлов**

Для удобства предусмотрена команда вывода текущего списка отслеживаемых файлов. Программа извлекает информацию из trackedFiles и отображает полный перечень путей. Это позволяет быстро убедиться, какие файлы находятся под наблюдением, и какие еще можно добавить.

Рисунок 6.3 – Пример отслеживаемых файлов

* 1. **Возможные ошибки и уведомления**

При попытке добавить уже отслеживаемый файл программа сообщает об этом и не выполняет повторное добавление.

При невозможности установить inotify-наблюдение (например, из-за отсутствия прав доступа) выводится соответствующее сообщение с расшифровкой ошибки.

Если при удалении файла он не найден в списке отслеживаемых, программа уведомит об этом.

Попытка начать мониторинг без добавленных файлов приведёт к сообщению о пустом списке.

При критических ошибках (например, недоступность дескриптора inotify) программа завершает выполнение с выводом причины.

Выход из программы

Выход осуществляется через отдельную команду в главном меню. При завершении мониторинга и работы с файлами все дескрипторы корректно закрываются, а структура отслеживания очищается.

**ЗАКЛЮЧЕНИЕ**

В рамках курсового проекта была разработана многопоточная программа для контроля изменений и проверки целостности группы файлов, обеспечивающая надежный мониторинг файловой системы в реальном времени. Решение направлено на минимизацию рисков несанкционированных изменений данных и оперативное выявление угроз, что особенно актуально в условиях роста требований к информационной безопасности.

Программа реализована как консольное приложение с поддержкой интерактивного интерфейса на базе библиотеки ncurses, что обеспечивает удобство взаимодействия без потери функциональности. Использование многопоточности (через std::thread и pthread) позволило организовать параллельную обработку событий файловой системы, что значительно повысило производительность и снизило задержки при работе с большими объемами данных.

Для контроля целостности файлов применены криптографические алгоритмы хеширования (SHA-256, BLAKE3), гарантирующие точность обнаружения изменений. Интеграция с системными API (inotify для Linux, FileSystemWatcher для Windows) обеспечила кросс-платформенность и эффективное отслеживание событий в реальном времени. Резервный механизм периодического сканирования добавил устойчивости к сбоям в работе API.

Особое внимание уделено отказоустойчивости:

- Транзакционное выполнение критических операций.

- Автоматическое создание резервных копий измененных файлов.

- Логирование событий в файлы (file\_monitor.log, changes.log) для последующего анализа.

- Использование мьютексов и семафоров для синхронизации потоков.

Программа прошла тестирование на различных сценариях, включая:

- Одновременный мониторинг 100+ файлов.

- Имитацию атак на целостность данных.

- Работу в фоновом режиме с минимальным потреблением ресурсов.

Результаты подтвердили высокую надежность и производительность решения. По сравнению с аналогами (Tripwire, AIDE) разработанная программа отличается гибкостью настройки, поддержкой фонового режима и упрощенным управлением через интерактивное меню.

Перспективы развития проекта включают:

* + - * Внедрение графического интерфейса (Qt, GTK).
      * Добавление сетевой синхронизации для распределенного мониторинга.
      * Интеграцию с облачными хранилищами для резервирования данных.
      * Поддержку дополнительных алгоритмов хеширования (SHA-3, Whirlpool).

Разработанное решение может быть адаптировано для использования в корпоративных системах аудита, системах резервного копирования и других сферах, где критически важна сохранность данных. Проект демонстрирует эффективное сочетание современных технологий (C++17, std::filesystem) и низкоуровневых подходов, что открывает возможности для его масштабирования и интеграции в комплексные системы безопасности.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] International Organization for Standardization. Programming languages — C++: ISO/IEC 14882:2020 / ISO. ‒ Geneva : ISO, 2020. ‒ 1870 p.

[2] Open Group. The Open Group Base Specifications Issue 7: IEEE Std 1003.1-2017 / The Open Group. ‒ Berkshire : The Open Group, 2018. ‒ 4032 p.

[3] Kerrisk, M. Linux Programmer’s Manual: inotify(7) / M. Kerrisk. ‒ 2023. ‒ Режим доступа: https://man7.org/linux/man-pages/man7/inotify.7.html. ‒ Дата обращения: 19.05.2024.

[4] Williams, A. C++ Concurrency in Action: Practical Multithreading / A. Williams. ‒ 2nd ed. ‒ Shelter Island : Manning Publications, 2019. ‒ 545 p.

[5] National Institute of Standards and Technology. Secure Hash Standard (SHS): NIST FIPS 180-4 / NIST. ‒ Gaithersburg, MD : NIST, 2015. ‒ 36 p.

[6] GNU Project. NCURSES Programming HOWTO [Электронный ресурс]. ‒ Режим доступа: https://tldp.org/HOWTO/NCURSES-Programming-HOWTO/. ‒ Дата обращения: 19.05.2024.

[7] Microsoft. C++ Standard Library: Filesystem / Microsoft Docs. ‒ 2023. ‒ Режим доступа: https://learn.microsoft.com/en-us/cpp/standard-library/filesystem. ‒ Дата обращения: 19.05.2024.

[8] Stack Overflow Community. Inotify Example Code [Электронный ресурс]. ‒ 2015. ‒ Режим доступа: https://stackoverflow.com/questions/7866535/inotify-example. ‒ Дата обращения: 19.05.2024.

[9] Google. Google Test: C++ Testing Framework [Электронный ресурс]. ‒ Режим доступа: https://github.com/google/googletest. ‒ Дата обращения: 19.05.2024.

[10] Cppreference Team. C++ reference [Электронный ресурс]. ‒ Режим доступа: https://en.cppreference.com/w/. ‒ Дата обращения: 19.05.2024.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Диаграмма классов**

**ПРИЛОЖЕНИЕ Б**

(обязательное)

**Схема структурная**

**ПРИЛОЖЕНИЕ В**

(обязательное)

**Схема алгоритма**

**ПРИЛОЖЕНИЕ Г**

(обязательное)

**Схема алгоритма**

**ПРИЛОЖЕНИЕ Д**

(обязательное)

**Ведомость документов**