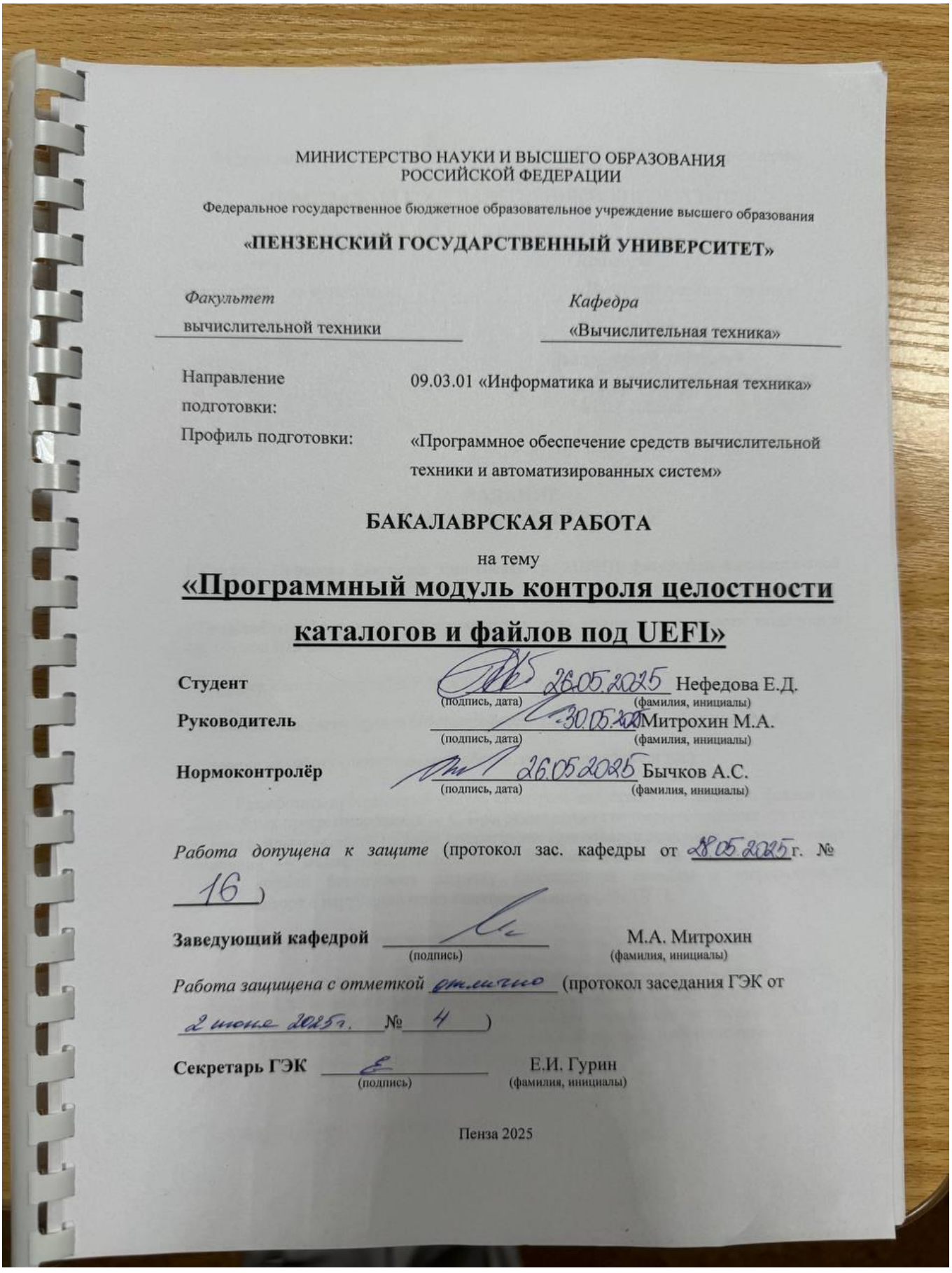


МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет вычислительной техники

Кафедра «Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

«Программный модуль контроля целостности каталогов и файлов под UEFI»

Студент 26.05.2025 Нефедова Е.Д.
(подпись, дата) (фамилия, инициалы)

Руководитель 30.05.2025 Митрохин М.А.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр 26.05.2025 Бычков А.С.
(подпись, дата) (фамилия, инициалы)

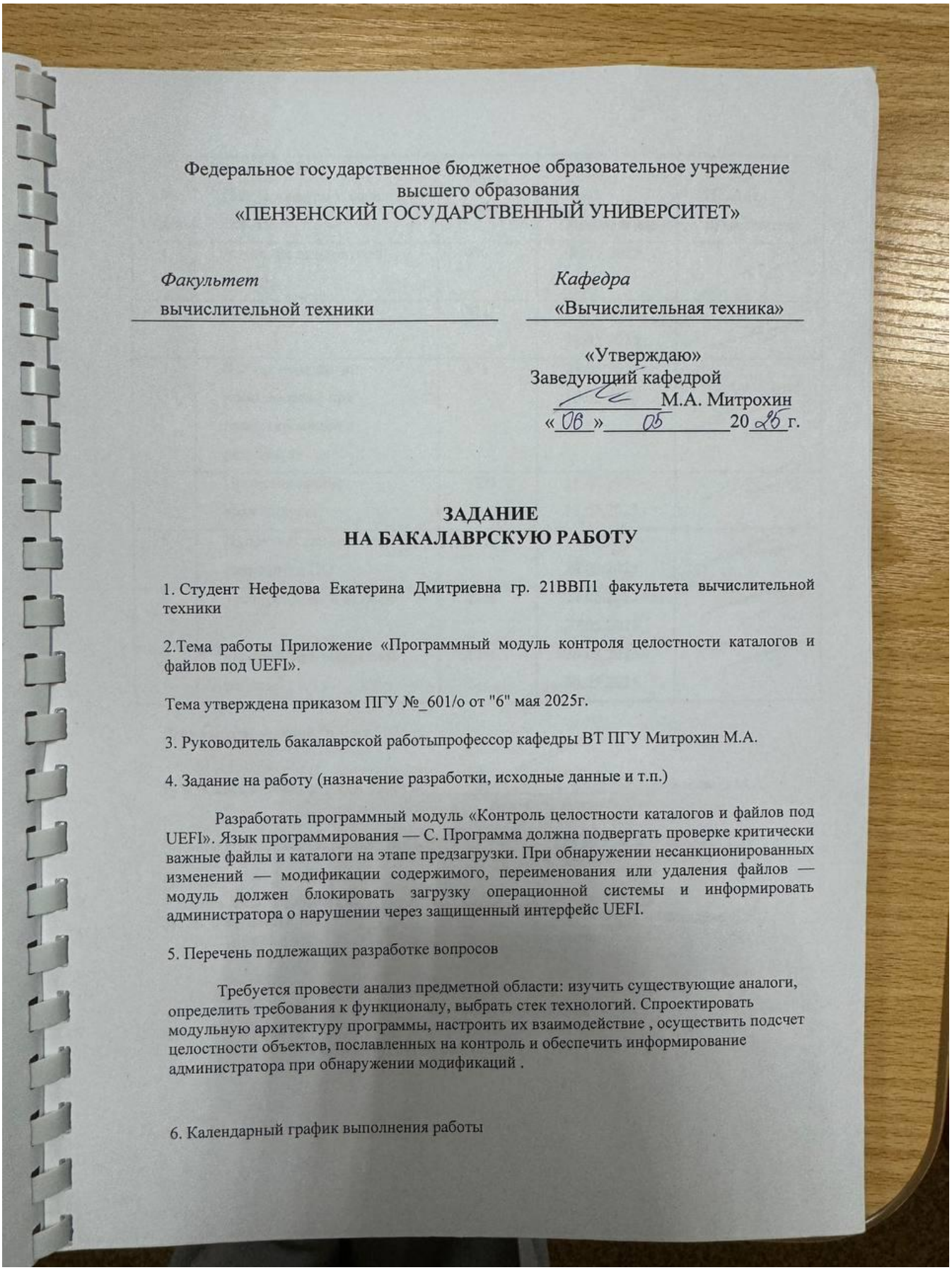
Работа допущена к защите (протокол зас. кафедры от 28.05.2025 г. № 16)

Заведующий кафедрой _____ М.А. Митрохин
(подпись) (фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от 2 июня 2025 г. № 4)

Секретарь ГЭК _____ Е.И. Гурин
(подпись) (фамилия, инициалы)

Пенза 2025



Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет вычислительной техники

Кафедра «Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«06» 05 2025 г.

ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Нефедова Екатерина Дмитриевна гр. 21ВВП1 факультета вычислительной техники

2.Тема работы Приложение «Программный модуль контроля целостности каталогов и файлов под UEFI».

Тема утверждена приказом ПГУ №_601/о от "6" мая 2025г.

3. Руководитель бакалаврской работыпрофессор кафедры ВТ ПГУ Митрохин М.А.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать программный модуль «Контроль целостности каталогов и файлов под UEFI». Язык программирования — С. Программа должна подвергать проверке критически важные файлы и каталоги на этапе предзагрузки. При обнаружении несанкционированных изменений — модификации содержимого, переименования или удаления файлов — модуль должен блокировать загрузку операционной системы и информировать администратора о нарушении через защищенный интерфейс UEFI.

5. Перечень подлежащих разработке вопросов

Требуется провести анализ предметной области: изучить существующие аналоги, определить требования к функционалу, выбрать стек технологий. Спроектировать модульную архитектуру программы, настроить их взаимодействие , осуществить подсчет целостности объектов, поставленных на контроль и обеспечить информирование администратора при обнаружении модификаций .

6. Календарный график выполнения работы

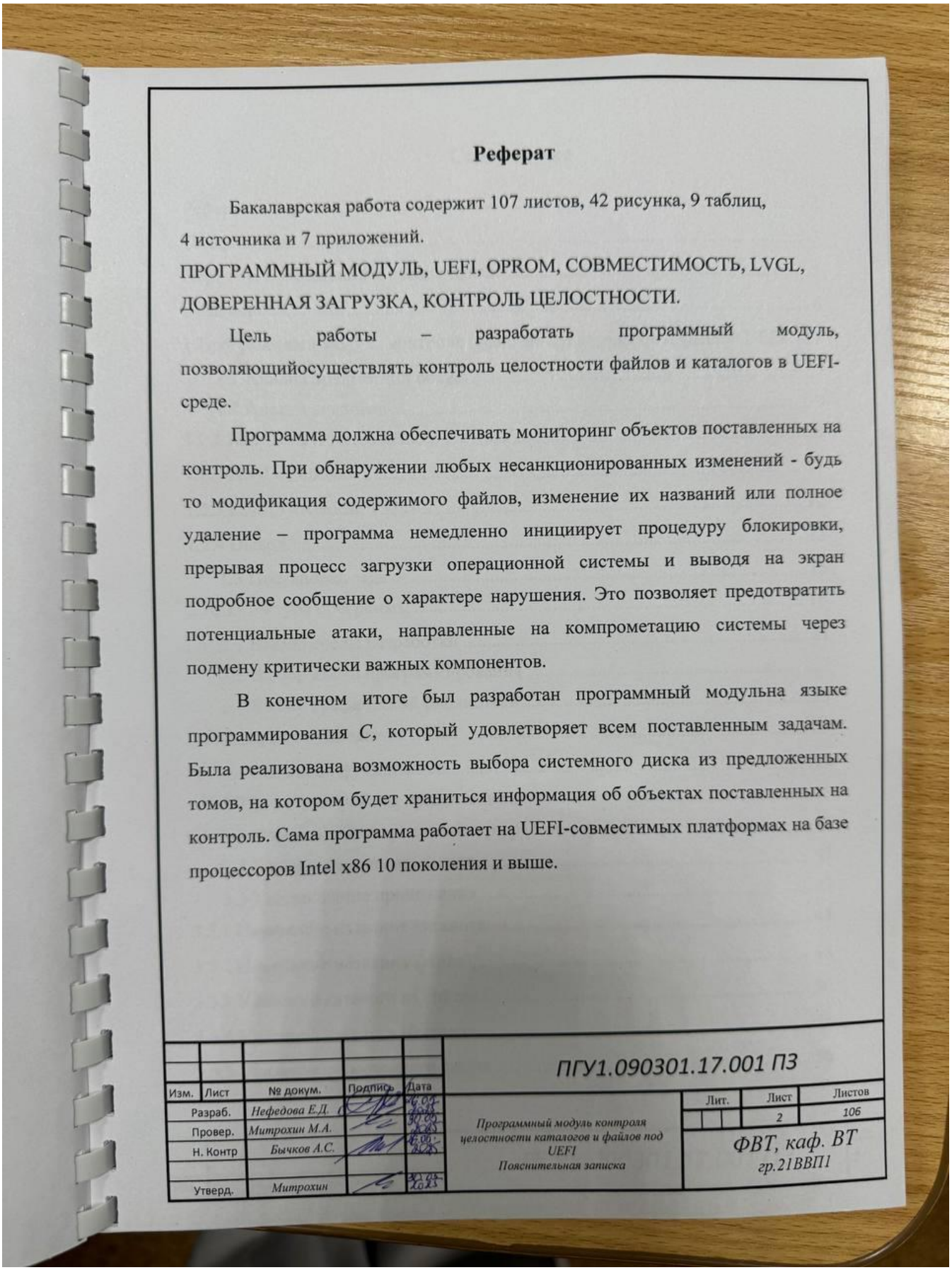
№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	4%	06.05.2025	
2	Анализ требований	8%	07.05.2025-08.05.2025	
3	Выбор технологий, применяемых при проектировании и разработке	8%	09.05.2025-10.05.2025	
4	Проектирование архитектуры	12%	11.05.2025-13.05.2025	
5	Проектирование и Разработка ПО	44%	14.05.2025-24.05.2025	
6	Тестирование и отладка	12%	25.05.2025-27.05.2025	
7	Оформление выпускной работы	12%	28.05.2025-30.05.2025	

Дата выдачи задания "6" мая 2025 г.

Руководитель бакалаврской работы _____ 06.05.2025 Митрохин М.А.
(подпись, дата)

Задание к исполнению принял студент _____ 06.05.2025 Нефедова Е.Д..
(подпись, дата)

Работу к защите допустить
Декан факультета _____ 30.05.2025 Фионова Л.Р.
(подпись, дата)



Реферат

Бакалаврская работа содержит 107 листов, 42 рисунка, 9 таблиц, 4 источника и 7 приложений.

ПРОГРАММНЫЙ МОДУЛЬ, UEFI, OPROM, СОВМЕСТИМОСТЬ, LVGL, ДОВЕРЕННАЯ ЗАГРУЗКА, КОНТРОЛЬ ЦЕЛОСТНОСТИ.

Цель работы – разработать программный модуль, позволяющийосуществлять контроль целостности файлов и каталогов в UEFI-среде.

Программа должна обеспечивать мониторинг объектов поставленных на контроль. При обнаружении любых несанкционированных изменений - будь то модификация содержимого файлов, изменение их названий или полное удаление – программа немедленно инициирует процедуру блокировки, прерывая процесс загрузки операционной системы и выводя на экран подробное сообщение о характере нарушения. Это позволяет предотвратить потенциальные атаки, направленные на компрометацию системы через подмену критически важных компонентов.

В конечном итоге был разработан программный модульна языке программирования *C*, который удовлетворяет всем поставленным задачам. Была реализована возможность выбора системного диска из предложенных томов, на котором будет храниться информация об объектах поставленных на контроль. Сама программа работает на UEFI-совместимых платформах на базе процессоров Intel x86 10 поколения и выше.

					ПГУ1.090301.17.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Нефедова Е.Д.			Программный модуль контроля целостности каталогов и файлов под UEFI Пояснительная записка	Лит.	Лист 2	Листов 106
Провер.		Митрохин М.А.						
Н. Контр		Бычков А.С.				ФВТ, каф. ВТ гр.21ВВП1		
Утверд.		Митрохин						

Содержание

Реферат 2
Содержание 3
Перечень сокращений и обозначений 9
Введение 10
1 Программный модуль контроля целостности каталогов и файлов UEFI 11
1.1 Анализ предметной области 11
1.2 Анализ аналогов 13
1.2.1 ПМДЗ «ViPNetSafeBoot» 13
1.2.2 ПАК «Соболь» 15
1.2.3 ПМДЗ «Aladdin TSM» 16
1.2.4 Вывод 17
1.3 Постановка задачи 19
2 Архитектура программы 20
2.1 Выбор среды разработки 25
2.2 Выбор языка программирования 26
2.3 Выбор графической библиотеки 27
3 Реализация приложения 29
3.1 Описание функционала 29
3.2 Описание работы приложения 37
3.3 Используемые протоколы при работе с файловыми системами 40
3.4 Руководство пользователя 41
3.5 Тестирование приложения 47
3.5.1 Изменение названия каталога 48
3.5.2 Изменение названия файла 52
3.5.3 Удаление каталога из списка 54
3.5.4 Удаление каталога из списка 57
3.5.5 Удаление содержимого файла 60

4 Безопасность жизнедеятельности 62
4.1 Правила и техника безопасности при работе с ПК.....62
4.2 Предварительная подготовка 63
4.3 Завершение работы с ПК.....64
Заключение 65
Список использованных источников.....66

Перечень сокращений и обозначений

В данной работе применяются следующие сокращения и обозначения:

АПМДЗ – аппаратно-программный модуль доверенной загрузки

ОС – операционнаясистема

ПК – персональный компьютер

IBMPC (International Business Machines Personal Computer) – первый массово производимый персональный компьютер

IDE (Integrated Development Environment) – интегрированная среда разработки

LVGL (Lightand Versatile Graphics Library) – графическая библиотека для создания интерфейса

MAC (Message Authentication Code) –специальный набор символов, который добавляется к сообщению и предназначен для обеспечения его целостности

PCI (Peripheral component interconnect) – шина ввода-вывода для подключения периферийных устройств к материнской плате компьютера

ROM (Read-OnlyMemory) – это тип компьютерной памяти, в которой хранятся постоянные данные и инструкции для запуска устройства

WPBT (Windows Platform Binary Table) – это таблица расширенной конфигурации и интерфейса питания в прошивке UEFI

Введение

Разработка приложений под UEFI является важным этапом создания современных технологических систем. UEFI (Unified Extensible Firmware Interface) является набором стандартов, заменяющих традиционную BIOS в компьютерах и других устройствах. Одним из основных преимуществ UEFI является его возможность загружать операционные системы и приложения прямо с жесткого диска или из сети, что обеспечивает быструю работу и высокую надежность.

Поддержка UEFI в разработке приложений открывает перед разработчиками широкие возможности для создания более современных, функциональных и безопасных систем. Она позволяет программистам использовать множество дополнительных возможностей, таких как защита от вредоносного ПО и улучшенный доступ к аппаратному обеспечению. Кроме того, UEFI обеспечивает более простой и гибкий интерфейс для взаимодействия между человеком и управляемым устройством.

Таким образом, разработка приложений под UEFI является важным шагом в создании современных технологических систем. Она позволяет повысить надежность, безопасность и производительность систем управления, обеспечивая более эффективное взаимодействие между человеком и машиной.

1 Программный модуль контроля целостности каталогов и файлов под UEFI

1.1 Анализ предметной области

В настоящее время производители материнских плат IBM *PC*-совместимых компьютеров осуществляют активный переход от устаревшего интерфейса BIOS к современной технологии UEFI. Это вызвано не только увеличением производительности вычислительных устройств, но и их распространенностью в различных областях.

Для обеспечения работы рабочей станции, корректной инициализации при старте операционной системы и дальнейшего ее функционирования необходима разработка специальных OptionROM (OpRom) и внедрение UEFI. OpROM представляет собой специализированный программный компонент, который используется для расширения функциональности базового программного обеспечения, отвечающего за начальную загрузку системы. Фазы работы BIOS представлены на рисунке 1.

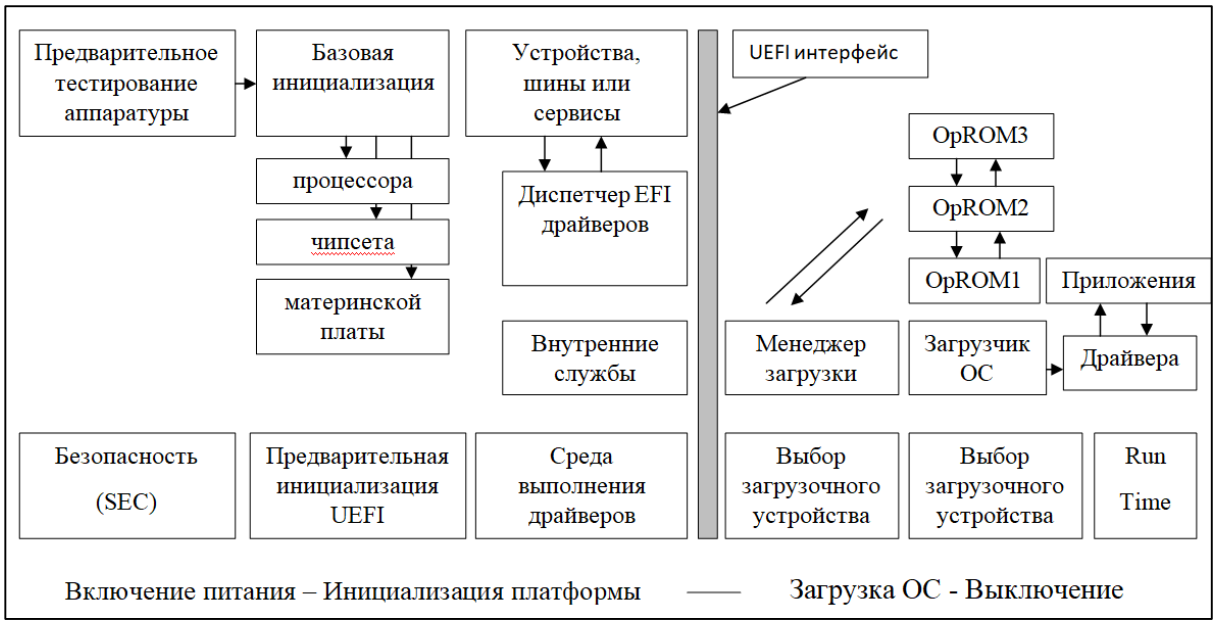


Рисунок 1 - Фазы работы BIOS

Учитывая разнообразие вычислительных устройств с точки зрения аппаратной конфигурации, разработка и использование OpRom становится крайне актуальной задачей. Наличие OpRom позволяет обеспечить совместимость вычислительных устройств с различным аппаратным обеспечением за счет корректной инициализации операционной системы, драйверов периферийных устройств.

Для обеспечения безопасного процесса загрузки операционной системы необходим модуль доверенной загрузки (МДЗ), который можно реализовать на основе архитектуры UEFI. Этот аппаратно-программный модуль служит для предотвращения несанкционированного доступа к информации разных уровней секретности и запускается на этапе инициализации системы. Если все компоненты безопасны, система продолжает загрузку и инициализацию оборудования. В случае успешной проверки всех модулей создается доверенная цепочка, что позволяет уверенно продолжить загрузку операционной системы. Рассмотрим основные этапы работы:

– осуществляется подготовка аппаратных ресурсов и корректная настройка оборудования (инициализация);

– контроль целостности – этот процесс позволяет убедиться в том, что программное обеспечение не было подвержено внешним воздействиям и соответствует исходным доверенным версиям, что является критически важным элементом для защиты от вредоносных программ;

– аутентификация пользователя запрашивает учетные данные, такие как логин и пароль, для подтверждения легитимности доступа к системе;

– разрешение на загрузку ОС, на данном этапе система принимает решение о том, разрешить или запретить загрузку операционной системы на основании успешно пройденных предыдущих этапов;

– журналирование и контроль событий, в процессе работы все действия, связанные с загрузкой и использованием устройства, фиксируются в журнале событий. это позволяет администратору или пользователю отслеживать все

попытки доступа или изменения, а также анализировать возможные угрозы безопасности.

В моей работе речь пойдет о написании программного модуля, отвечающего за одну из ключевых функций МДЗ, описанных выше, а именно - контроль целостности файлов и каталогов.

1.2 Анализ аналогов

В процессе разработки нового программного продукта важно учитывать существующие решения на рынке. Анализ аналогов позволяет не только оценить актуальные продукты на рынке, но и выявить их преимущества и недостатки. В рамках разработки были проанализированы уже существующие МДЗ:

– *ViPNetSafeBoot*;

– ПАК «Соболь»;

– АПМДЗ «Криптон-замок».

Проведен анализ и выявлены слабые и сильные стороны каждого из приложений.

1.2.1 ПМДЗ «ViPNetSafeBoot»

Интерфейс представлен на рисунке 2.

Достоинства:

– ViPNetSafeBoot блокирует запись malware в UEFI BIOS, с последующим его размещением в WPBT (Windows Platform Binary Table) во время загрузки ОС, тем самым предотвращая запись вредоносного кода в WPBT во время загрузки ОС;

– неизвлекаемость обеспечивает невозможность обхода модуля даже при наличии физического доступа к компьютеру или серверу;

– программные модули доверенной загрузки в BIOS запускаются раньше, чем ПО из ROM плат расширения, что позволяет обеспечивать защиту от вредоносного ПО на более раннем этапе.

Недостатки:

– поддержка аппаратных токенов узкой линейки продуктов. Не поддерживаются устройства eSmartToken, SafeTech и другие известные производители;

– ограничения по поддержке операционных систем. Функция автоопределения компонентов загрузки операционной системы для постановки на контроль целостности доступна только для Windows;

– сложности с настройкой некоторых функций. Не все функции в интерфейсе понятны интуитивно.

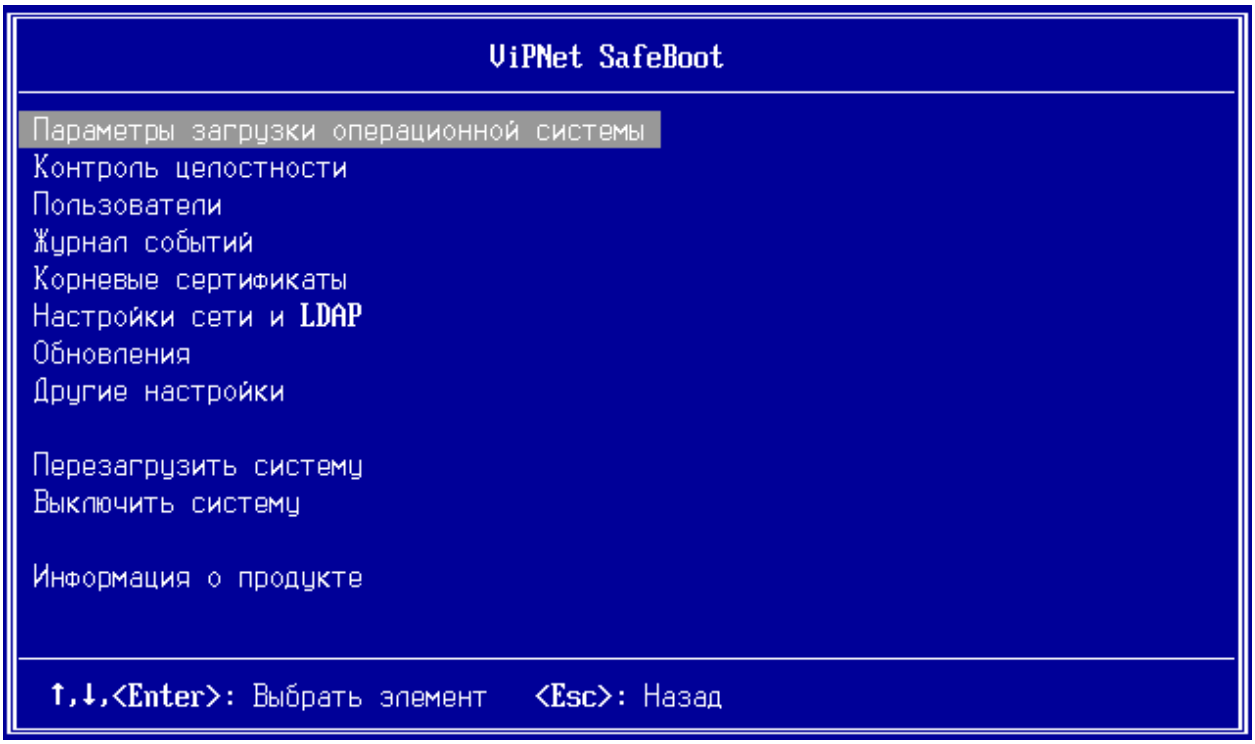


Рисунок 2 - Внешний вид меню режима администрирования ViPNet SafeBoot

1.2.2 ПАК «Соболь»

Интерфейс представлен на рисунке 3.

Достоинства:

– Возможность программной инициализации без вскрытия системного блока, централизованное управление пользователями через SecretNetStudio;

– Поддержка широкого списка ОС семейства Windows и Linux.

Недостатки:

– Не поддерживается контроль целостности секторов, расположенных на диске за пределами 2 Тбайт;

– Не поддерживается контроль целостности файлов, расположенных на динамических дисках.

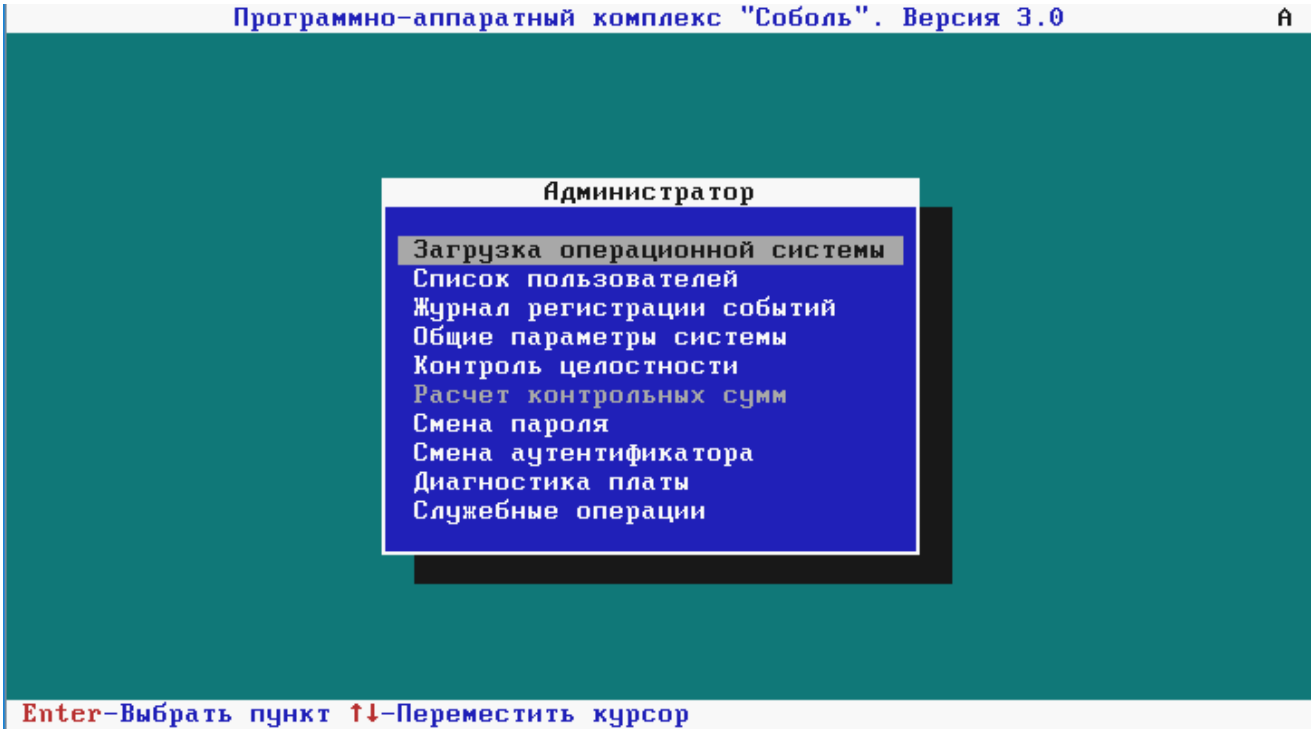


Рисунок 3 – Меню администратора

1.2.3 ПМДЗ «Aladdin TSM»

Интерфейс представлен на рисунке 4.

Достоинства:

– дистанционное управление ключами и сертификатами;

– контроль над коммуникациями и процессами загрузки. Aladdin TSM позволяет контролировать взаимодействие с любой «гостевой» системой без необходимости изменений в ней или используемых приложениях.

Недостатки:

– необходимость разбиения операционной системы на разделы R/O и R/W. Это нужно для контроля файловой системы;

– возможная утрата доступа к управлению. Доступ может быть утрачен, если забыт пароль администратора, утерян токен для доступа, учётная запись удалена другим пользователем с правами администратора и так далее.

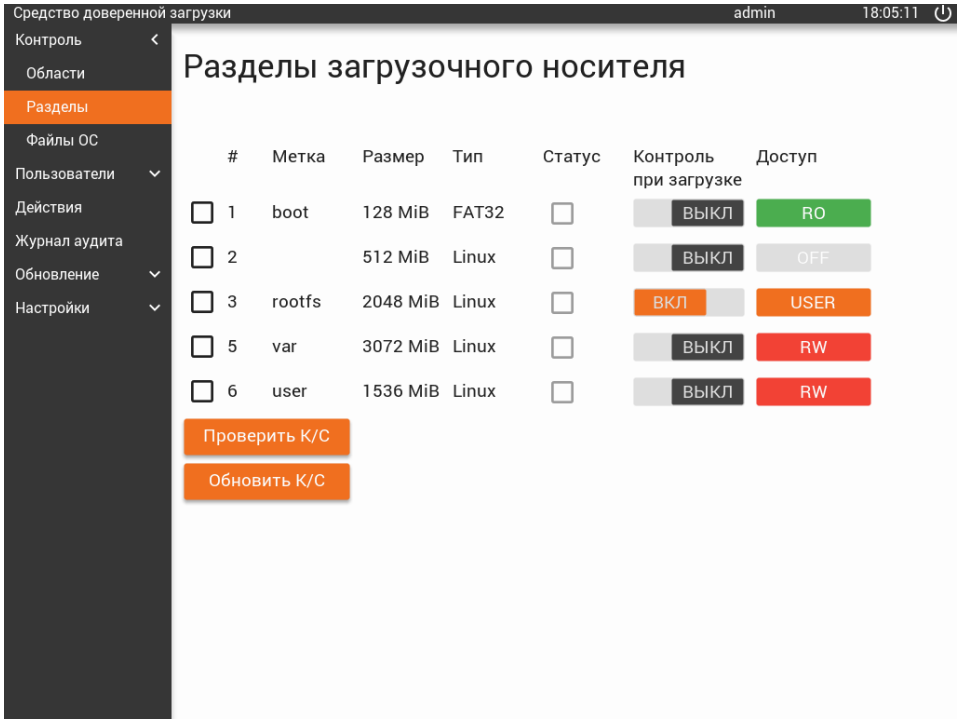


Рисунок 4 - Интерфейс Aladdin TSM. Режим администрирования

1.2.4 Вывод

В процессе исследования существующих аналогов было обнаружено, что ни один из них не обладает полным набором необходимых функций. В таблице 1 приведены основные критерии анализа приложений-аналогов.

Таблица 1 – Достоинства и недостатки приложений-аналогов

	ПМДЗ «VipNetSafeBoot»	ПАК «Соболь 4.2»	ПМДЗ «AladdinTSM»
Исполнение продукта	Программный модуль	Плата расширения	Программный модуль
Поддержка UEFI	Да	Да	Да
Требования к аппаратному обеспечению	Дополнительных требований к аппаратному обеспечению нет	UEFI (вер. 4.2), архитектура x86	Материнская плата с объёмом свободной встроенной памяти не менее 1 МБ
Поддерживаемые идентификаторы	USB-токенJaCarta	JaCarta PKI, ГОСТ (USB-носитель и смарт-карта), JaCarta PKI/ГОСТ, JaCarta-2 ГОСТ (USB-	USB-ключи eToken PRO, eToken PRO (Java), Rutoken, Rutoken RF, Rutoken ЭЦП, RutokenLite,

	ПМДЗ «VipNetSafeBoot»	ПАК «Соболь 4.2»	ПМДЗ «AladdinTSM»
		носитель и смарт-карта), Рутокен ЭЦП, ЭЦП 2.0 (USB-носитель и смарт-карта), РутокенLite, 2151, ЭЦП PKI (смарт-карта), eTokenProJava, SafeNeteToken 5100, SafeNeteToken 5105, 5200, 5205	JaCarta-2 ГОСТ, JaCarta-2 PKI/ГОСТ, JaCarta SF/ГОСТ, Guardant-ID, смарт-карты eToken PRO, eToken PRO (Java), Rutoken ЭЦП, RutokenLite, JaCarta-2 ГОСТ, JaCarta-2 PKI/ГОСТ, электронная карта

Сравнительный анализ МДЗ, представленных на рынке, выявил высокую конкуренцию между существующими системами. При этом, учитывая преимущества и ограничения каждой из них, была подтверждена одна из ключевых тенденций сегодняшнего дня – переход на UEFI. Поддержка UEFI в разработке МДЗ не просто актуальна, а становится необходимостью для обеспечения надежной и безопасной загрузки оборудования. Такой подход

позволяет реализовать более совершенные механизмы защиты, которые соответствуют современным требованиям безопасности.

1.3 Постановка задачи

Задача, которую предстоит решить, заключается в разработке эффективного модуля контроля целостности каталогов и файлов, интегрированного в среду UEFI. Основная цель такого модуля заключается в обеспечении защиты системы на самом раннем этапе её загрузки, что является критически важным для предотвращения атак, направленных на подрыв функционирования операционной системы. Программа должна быть способна эффективно проверять целостность каждого файла и каталога, чтобы в случае обнаружения несоответствия – инициировать процедуру блокировки загрузки и сигнализировать о возможной угрозе безопасности.

Важно также предусмотреть простоту настройки и управления модулем. Пользователь или администратор системы должны иметь возможность легко настраивать параметры контроля целостности, включая выбор файлов и каталогов, подлежащих проверке, а также действия, которые должны быть предприняты в случае выявления нарушений.

В результате, реализованный программный модуль контроля целостности не только повысит уровень безопасности загружаемых систем, но и станет важным элементом системы защиты информации, способствуя формированию защищенной и надежной вычислительной среды.

1.4 Требования к аппаратуре и программному обеспечению

Для корректной работы программного модуля, должны быть соблюдены требования к аппаратуре и ПО, а именно:

– ОС семейства Windows;

– UEFI совместимая платформа на базе процессоров Intel x86 10 поколения и выше.

2 Архитектура программы

Архитектура средств доверенной загрузки (СДЗ) строится на принципах поэтапной верификации программного обеспечения, формируя непрерывную цепочку доверия от аппаратного уровня до загрузки операционной системы.

Нормативной основой для разработки архитектуры СДЗ служит информационное письмо ФСТЭК России от 06.02.2014 г. № 240/24/405 «Об утверждении Требований к средствам доверенной загрузки», в котором определены три ключевых уровня реализации механизмов доверенной загрузки:

- средства доверенной загрузки уровня базовой системы ввода-вывода;
- средства доверенной загрузки уровня платы расширения;
- средства доверенной загрузки уровня загрузочной записи.

Реализованный программный модуль контроля целостности файлов и каталогов является частью МДЗ типа платы расширения. Средства доверенной загрузки уровня платы расширения – это аппаратно-программные СДЗ (или модули АПМДЗ), которые реализуются в виде отдельной платы расширения стандартов PCI, PCI Express, mini-PCI Express, M.2. Такие платы устанавливаются внутри устройства (сервера, ноутбука) и перехватывают управление порядком загрузки устройства, исключая загрузку ОС с внешних накопителей (флешек, CD/DVD-дисков, сетевых расположений).

После включения компьютера начинается многоэтапный процесс инициализации системы. На начальном этапе BIOS/UEFI в ходе процедуры POST идентифицирует все подключенные устройства, включая плату расширения с модулем доверенной загрузки. Обнаружив такую плату, система распознает ее как приоритетное устройство безопасности и начинает процесс инициализации. Первоначально выполняется проверка целостности самой платы – верифицируется контрольная сумма ее прошивки и аутентичность цифровой подписи с использованием криптографических алгоритмов,

соответствующих требованиям регуляторов. Только после успешного прохождения этих проверок управление последовательно передается на блок АПМДЗ (рисунок 5), который активирует свои защитные механизмы, в состав которых входит контроль целостности.

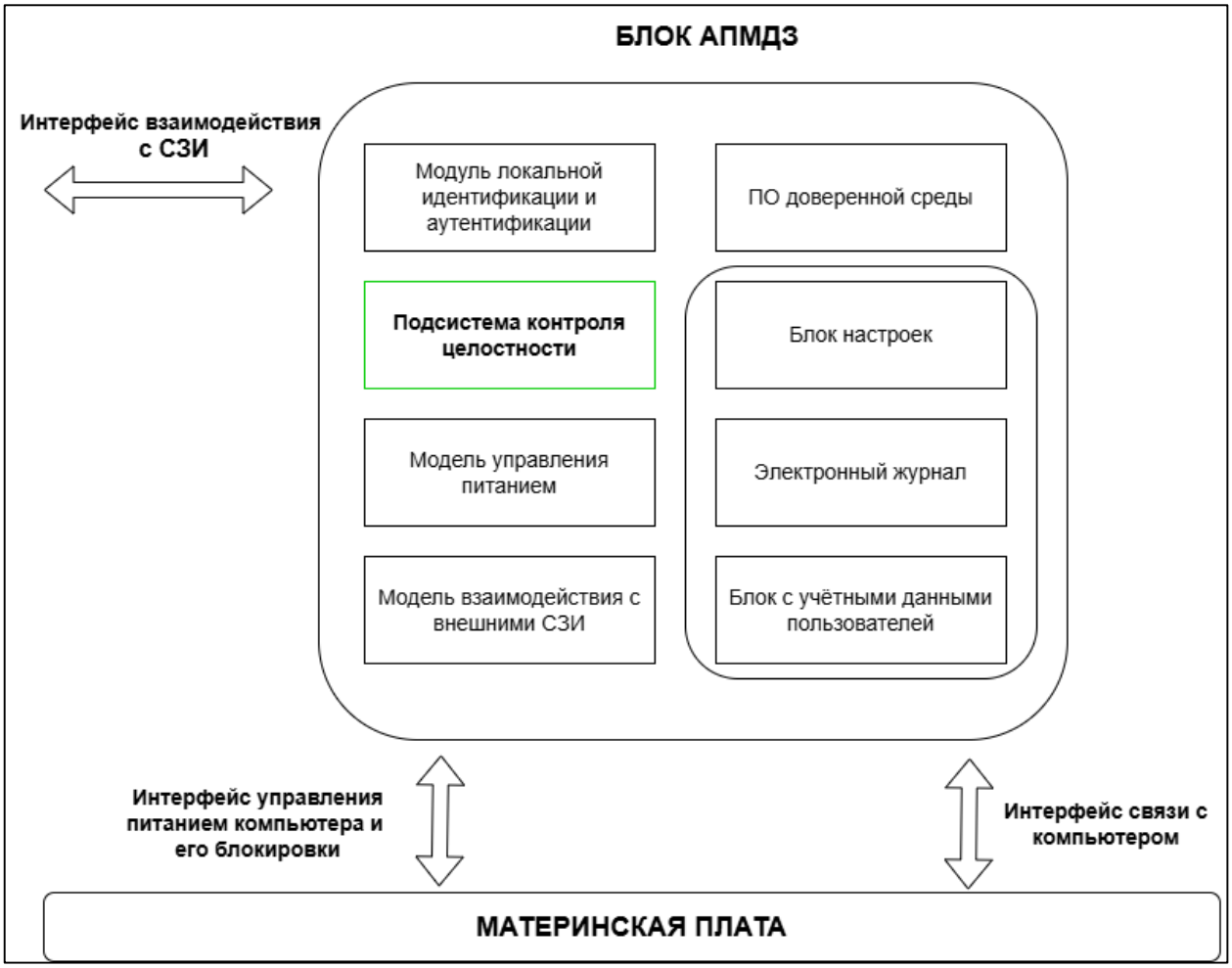


Рисунок 5 – Структурная схема блока АПМДЗ

Программное решение представляет собой модульную систему (схема архитектуры представлена на рисунке 6), главным узлом которой является проект AuthorizationDxe, содержащий основную точку входа в приложение. Он отвечает за последовательный вызов подпрограмм модулей на всех этапах доверенной загрузки. В данном случае рассматривается сценарий использования модуля подсчета контроля целостности файлов и каталогов.

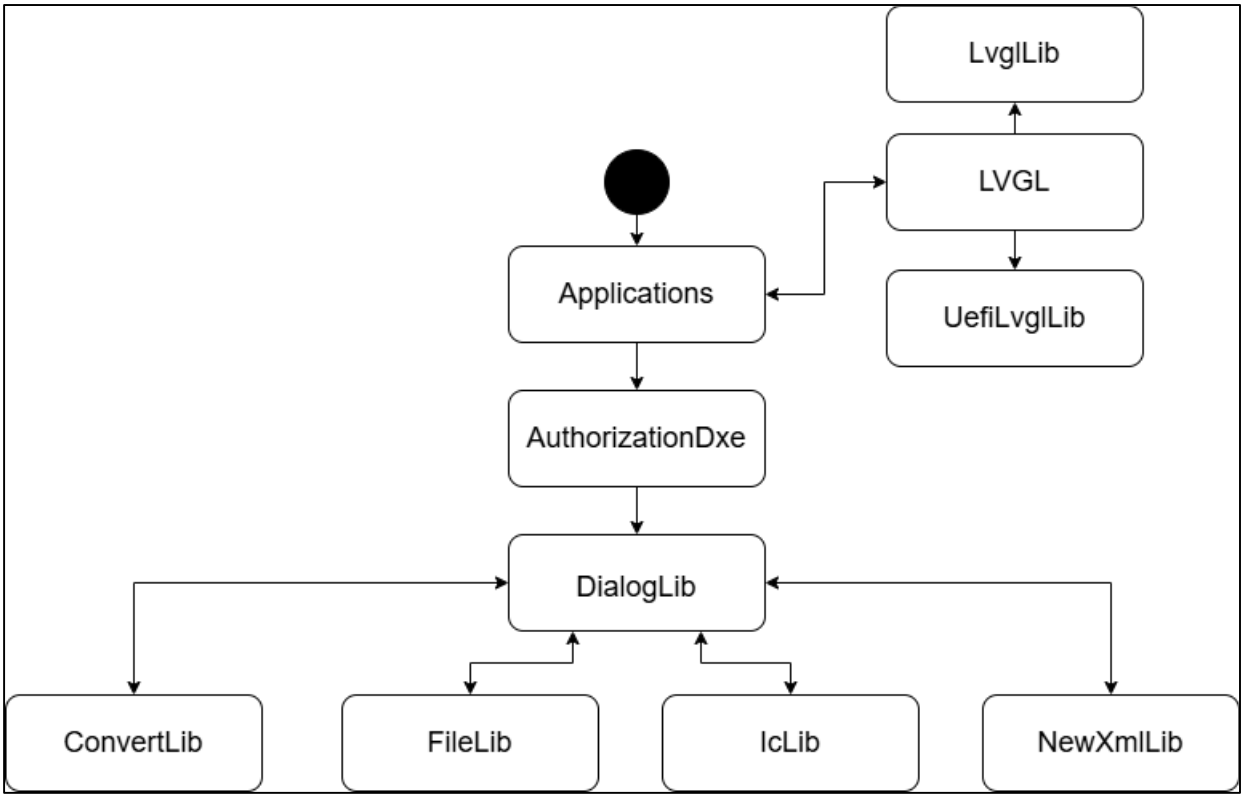


Рисунок 6 – Схема взаимодействия модулей архитектуры готового приложения

Поскольку разработанное ПО предназначено для использования в UEFIсреде, в качестве инструмента для создания пользовательского интерфейса использовалась библиотека LVGL – на самой начальной стадии были подключены соответствующие компоненты (UefiLvglLib).

Для обеспечения структурированности и удобства разработки был созданпакет проектов DialogLib, содержащийпроекты, каждый из которых отвечает за отдельные фазы проверок. Такая организация кода позволяет легко масштабировать систему, добавляя новые модуля для проведения дополнительных этаповпроверки без необходимости переписывания основного кода. Когда модуль AuthorizationDxe вызывает определенную функцию, управление передается соответствующему модулю из DialogLib, который выполняя свою задачу - возвращает управление обратно в основной узел, в данном случае управление передается модулю контроля целостности.

Для удобной работы с файловыми системами, информация о них представлена в виде структур efs_info* (таблица 2), tree_value_* (таблица3), directory_info_* (таблица 4).

Таблица 2 – Описание структуры efs_info*

Входной параметр	Тип данных	Назначение
pVolumeInfo	EFI_FILE_SYSTEM_INFO*	атрибуты тома
pVolumeInfoEx	FULL_VOLUME_INFO*	тип файловой системы, гуид раздела, гуид диска, тип разметки диска, серийный номер
fs_index	UINTN	индекс файловой системы
fileList	CHAR16*	список с файлами
fileListSize	UINTN	размер списка файлов
dirList	CHAR16*	список директорий
dirListSize	UINTN	размер списка директорий
root	tree_value_*	указатель на том (корень файловой

Входной параметр	Тип данных	Назначение
		системы)

Таблица 3 – Описание структуры tree_value_*

Входной параметр	Тип данных	Назначение
parent	tree_value_*	указатель на родителя
next	tree_value_*	указатель на следующий элемент
first_child	tree_value_*	указатель на первый дочерний элемент
type	int	тип объекта: директория (0), файл (1)
efs	efs_info_*	структура с информацией о томе
name	CHAR16*	имя объекта
Full_name;	CHAR16*	полный путь, включая имя объекта

Таблица 4 – Описание структуры directory_info_*

Входной параметр	Тип данных	Назначение
pefi_info	efs_info_*	указатель на структуру с информацией о томе
parent_directory	void*	указатель на родительскую директорию
dir_name	CHAR16*	имя директории

В процессе реализации были задействованы различные низкоуровневые механизмы UEFI, включая дисковые и файловые протоколы, протоколы PCI-устройств, а также функции работы с памятью. Для обеспечения взаимодействия с этими компонентами были разработаны специализированные проекты, каждый из которых содержит набор функций по назначению. Для работы с преобразованием типов, форматами представления и стилями использовалась библиотека ConvertLib. Взаимодействие с файловыми системами осуществляется с помощью FileLib. IcLib служит для хранения функций подсчета контрольных сумм, а также работу с файлом, содержащим результат. Для формирования xml-документа организован NewXmlLibпроект, в котором прописаны варианты взаимодействия с ним.

2.1 Выбор среды разработки

В качестве среды разработки будет использоваться Visual Studio.Visual Studio — это мощная интегрированная среда разработки (IDE), созданная

компанией Microsoft, которая предоставляет разработчикам все необходимые инструменты для создания программного обеспечения, включая приложения для UEFI, что позволяет разработчикам создавать совместимые и переносимые решения.

Платформа предлагает удобный интерфейс и широкий набор функций, которые значительно упрощают процесс написания, отладки и тестирования кода. Visual Studio поддерживает различные расширения и плагины, которые могут улучшить функциональность среды разработки, добавляя новые инструменты и библиотеки, необходимые для работы с UEFI.

Одной из ключевых возможностей, которые предоставляет Visual Studio, является использование эмуляторов. Это значительно упрощает процесс отладки, так как команды могут выявлять и исправлять ошибки без необходимости перезагружать физические устройства, что экономит время и повышает общую эффективность работы.

Использование Visual Studio для разработки приложений на UEFI может существенно увеличить производительность команды, благодаря предоставляемым инструментам, поддержке лучших практик и возможности легко интегрироваться с существующими рабочими процессами.

2.2 Выбор языка программирования

Для реализации данного проекта был выбран язык C, созданный в начале 1970-х годов и быстро завоевавший признание благодаря своей гибкости, эффективности и низкому уровню абстракции, что позволило программистам писать код, максимально близкий к железу. Одной из ключевых областей применения языка C является разработка программного обеспечения для встроенных систем и, в частности, прошивок UEFI.

Поскольку проект напрямую взаимодействует с UEFI-средой-критически важна производительность, позволяя разработчикам писать высокоэффективный код, который может работать близко к аппаратному

уровню, что критически важно для системного программирования, где скорость и эффективность имеют первостепенное значение, а также прямой доступ к аппаратным ресурсам, Спредоставляя возможность работы с указателями и прямого доступа к памяти, что необходимо для взаимодействия с аппаратными компонентами на уровне прошивки, что особенно важно в контексте UEFI, где требуется управление различными устройствами и ресурсами системы. Это именно тот уровень контроля, который необходим.

Не менее важно, что выбранная библиотека LVGL, которая является ключевой для реализации графического интерфейса, изначально написана на C и оптимально работает именно в этой экосистеме. Использование других языков потребовало бы создания сложных оберток и могло бы привести к потере производительности, что совершенно неприемлемо для встраиваемой системы.

2.3 Выбор графической библиотеки

В качестве графической библиотеки при написании модуля была выбрана LVGL (Lightand Versatile Graphics Library), известная своей легкостью и универсальностью, представляет собой открытую графическую библиотеку, предназначенную для создания встраиваемых графических интерфейсов. Она обладает рядом преимуществ, включая минимальные требования к ресурсам, что делает её идеальным выбором для устройств с ограниченными вычислительными возможностями. LVGL поддерживает широкий спектр виджетов и инструментов для разработки, позволяя создавать интуитивно понятные и визуально привлекательные пользовательские интерфейсы.

LVGL может использоваться для создания графических интерфейсов в приложениях UEFI, которые традиционно имели ограниченные возможности в плане визуализации. UEFI, являясь современным стандартом прошивки, заменяющим устаревший BIOS, открывает новые возможности для

разработчиков. Использование LVGL в контексте UEFI позволяет разработчикам создавать более дружелюбные и функциональные графические интерфейсы, такие как настройки BIOS или менеджеры загрузки, которые значительноулучшают взаимодействие пользователя с системой.

Применение LVGL для создания приложений под UEFI имеет несколько преимуществ. Во-первых, LVGL обладает высокой скоростью отрисовки и минимальным потреблением памяти, что важно для встраиваемых систем. Это позволяет создавать графические интерфейсы с плавными анимациями и быстрым откликом на действия пользователя.

Во-вторых, LVGL поддерживает различные типы ввода, включая сенсорные экраны, клавиатуры и даже голосовое управление. Это делает его универсальным инструментом для разработки интерфейсов, которые могут быть легко адаптированы под разные типы устройств.

Кроме того, LVGL имеет активное сообщество разработчиков, которые постоянно работают над улучшением библиотеки и добавлением новых функций. Это обеспечивает поддержку и дальнейшее развитие проектов, созданных с использованием LVGL.

Применение LVGL в UEFI-приложениях также способствует улучшению безопасности и надежности системы, поскольку она позволяет более четко визуализировать критические системные сообщения и предупреждения. Это особенно важно в контексте предзагрузочных приложений, где необходимо отслеживать и управлять журналами событий и ошибок. Таким образом, интеграция LVGL с UEFI открывает новые горизонты для разработки встраиваемых систем, делая их более доступными, безопасными и удобными для конечных пользователей. В целом, LVGL вносит значительный вклад в развитие интерфейсов встраиваемых систем, предоставляя разработчикам инструменты для создания высококачественных и адаптивных графических интерфейсов.

3 Реализация приложения

3.1 Описание функционала

Программный модуль доверенной загрузки обладает модульной структурой, что означает, что он состоит из отдельных взаимосвязанных компонентов или блоков, каждый из которых отвечает за выполнение определенных функций. Такая структура обеспечивает гибкость и удобство в разработке, тестировании и сопровождении системы, так как каждый компонент можно разрабатывать и отлаживать независимо, а также легко заменять или обновлять без необходимости внесения изменений во всю систему. Функциональная схема представлена на рисунке 7.

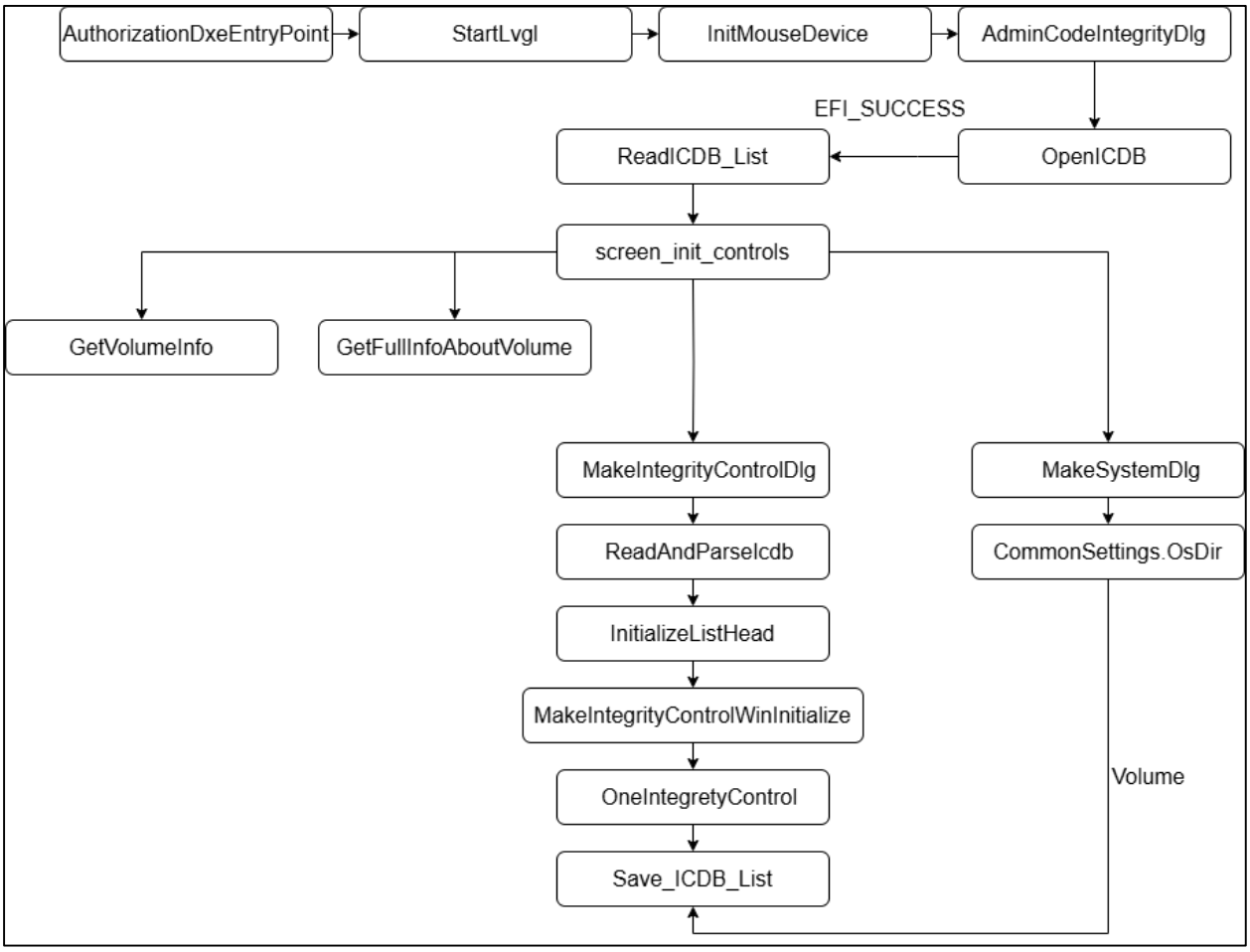


Рисунок 7 – Функциональная схема

Поскольку компонент подсчета целостности файлов и каталогов является составным звеном, чтобы интегрировать этот модуль в полноценную систему, необходимо было разработать основной проект, который бы служил точкой входа и моделировал работу всей системы в целом, как она могла бы выглядеть в реальной, крупной среде. Созданный мною проект под названием AuthorizationDxe является основным компонентом, из которого стартует выполнение всей системы доверенной загрузки.Главной функцией в этом проекте является AuthorizationDxeEntryPoint(),и именно отсюда инициируются вызовы различных подпрограмм и модулей, отвечающих за конкретные этапы проверки и верификации.

Разработка была ориентирована на работу в среде UEFI, поэтому процесс ее выполнения практически полностью основан на использовании соответствующих протоколов, что обусловило необходимость выполнения ряда предварительных подготовительных манипуляций перед непосредственным написанием функционала.

Для инициализации графической библиотеки LVGL, которая обеспечивает создание пользовательских интерфейсов в условиях низкоуровневого программирования и ограниченных ресурсов UEFI была использована функция StartLvgl(). Вызов этой функции позволяет подготовить необходимые компоненты для отображения графики, настроить дисплейные драйверы и запустить цикл обработки событий, обеспечивая взаимодействие с пользователем через графические элементы. Обычно она включает в себя настройку дисплея, регистрацию обратных вызовов, обработку таймеров и запуск основного цикла обработки задач графической системы.

Для более удобного взаимодействия с системой написана функция InitMouseDevice(), которая отвечает за инициализацию мыши. В среде UEFI для работы с указателем мыши и отображения графической информации широко используют протоколы EFI_SIMPLE_POINTER_PROTOCOL, EFI_SIMPLE_POINTER_STATE. Протокол

EFI_SIMPLE_POINTER_PROTOCOL обеспечивает доступ к данным о перемещениях и кликах указателя мыши, позволяя получать текущие состояния указателя через его методы. Для этого используют структуру EFI_SIMPLE_POINTER_STATE, в которой содержится информация о текущем положении курсора (AbsoluteX и AbsoluteY), а также о перемещениях (DeltaX и DeltaY) и событиях нажатия кнопок мыши. Эти данные необходимы для определения текущего положения указателя и реакции интерфейса на пользовательский ввод.

После успешного подключения необходимых для работы инструментов и компонентов системы, обеспечивает полноценную подготовку к дальнейшим операциям, происходит вызов программного модуля, ответственного за проверку целостности файлов и каталогов в разделе под названием AdminCodeIntegrityDlg(). Этот модуль предназначен для проведения диагностической проверки и обеспечения целостности данных, хранящихся в системе. В первую очередь в рамках этого процесса осуществляется попытка открытия файла ICDB, который содержит необходимую информацию о состоянии файлов и каталогов. При этом используется функция OpenICDB(), которая либо открывает существующий файл, либо создает новый — пустой, если файл отсутствует. После этого, если операция завершилась успешно и статус выполнения соответствует значению EFI_SUCCESS, что означает успешное выполнение команды и возможность получения доступа к содержимому файла, происходит вызов функции ReadICDB_List(). Эта функция предназначена для чтения файла ICDB.xml и инициализации глобальной структуры XML документа.

После успешной отработки подготовительных функций происходит вывод пользовательского интерфейса и заполнение его необходимой файловой информацией, которое происходит в функции screen_init_controls(). Этот этап является важным для обеспечения удобного и понятного взаимодействия пользователя с системой, поэтому особое внимание уделяется правильному

формированию и отображению элементов интерфейса, а также актуальных данных о файловых системах. Для работы с файловыми системами и их содержимым использовались специализированные функции, такие как GetVolumeInfo() и GetFullInfoAboutVolume(). Первая функция, GetVolumeInfo(), предназначена для получения базовой информации о разделе, включая его метки, размер, свободное место, а также список каталогов и файлов, которые он содержит. Эта функция позволяет определить характеристики раздела, его структуру и текущий статус, что является необходимым для отображения точных данных пользователю. Вторая функция, GetFullInfoAboutVolume(), обеспечивает более подробный сбор информации о файловой системе, включая ее тип, параметры форматирования, состояние, а также дополнительные метаданные, которые могут потребоваться для более полного отображения информации или для дальнейшей обработки.

Файловая информация представляет собой древовидную структуру:

```
Structtree_value_;
Typedefstructefs_info_
{
 EFI_FILE_SYSTEM_INFO* pVolumeInfo;
 FULL_VOLUME_INFO* pVolumeInfoEx;
 UINTN fs_index;
 CHAR16* fileList;
 UINTN fileListSize;
 CHAR16* dirList;
 UINTN dirListSize;
 Structtree_value_* root;
} efs_info;



Typedefstructtree_value_
{
 Structtree_value_* parent;
 Structtree_value_* next;
 Structtree_value_* first_child;
 int type;
 structefs_info_* efs;
 CHAR16* name;
 CHAR16* Full_name;
} tree_value;
```

```
typedefstructdirectory_info_
{
 efs_info* pefi_info;
 void* parent_directory;
 CHAR16* dir_name;

} direct_info;
```

Основная структура — это efs_info, которая представляет информацию о файловой системе или разделе. Она содержит ссылки на связанные объекты, такие как список дочерних элементов (child_efs_info_list), указатели на дополнительную информацию о томе (pVolumeInfo, pVolumeInfoEx), а также метаданные, такие как имя тома (szVolumeName), индекс файловой системы (fs_index), и списки файлов и директорий (fileList, dirList) с их размерами (fileListSize, dirListSize). Для представления дерева используется структура tree_value, которая содержит указатели на родителя (parent), следующего элемента (next), первого ребёнка (first_child), а также тип элемента (type), ссылку на связанный объект efs_info (efs) и имена (name, Full_name). В дополнение есть структура directory_info, связывающая информацию о конкретной директории с соответствующими данными (pefi_info) и именами. В целом структура данных организует информацию о файловых системах, их разделах и содержимом в иерархической форме с возможностью навигации по дереву и связным спискам.Наглядное представление взаимодействия между элементами изображено на рисунке 8.

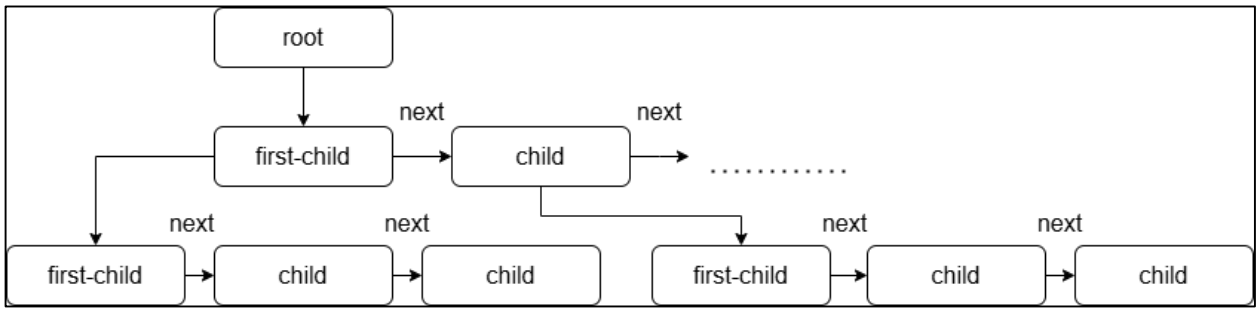


Рисунок 8 – Схема взаимодействия внутри файловой системы

После постановки файлов на контроль вызывается функция MakeIntegrityControlDlg(), которая выполняет роль запуска процесса проверки целостности системы. Алгоритм работы функции представлен на рисунке 9.

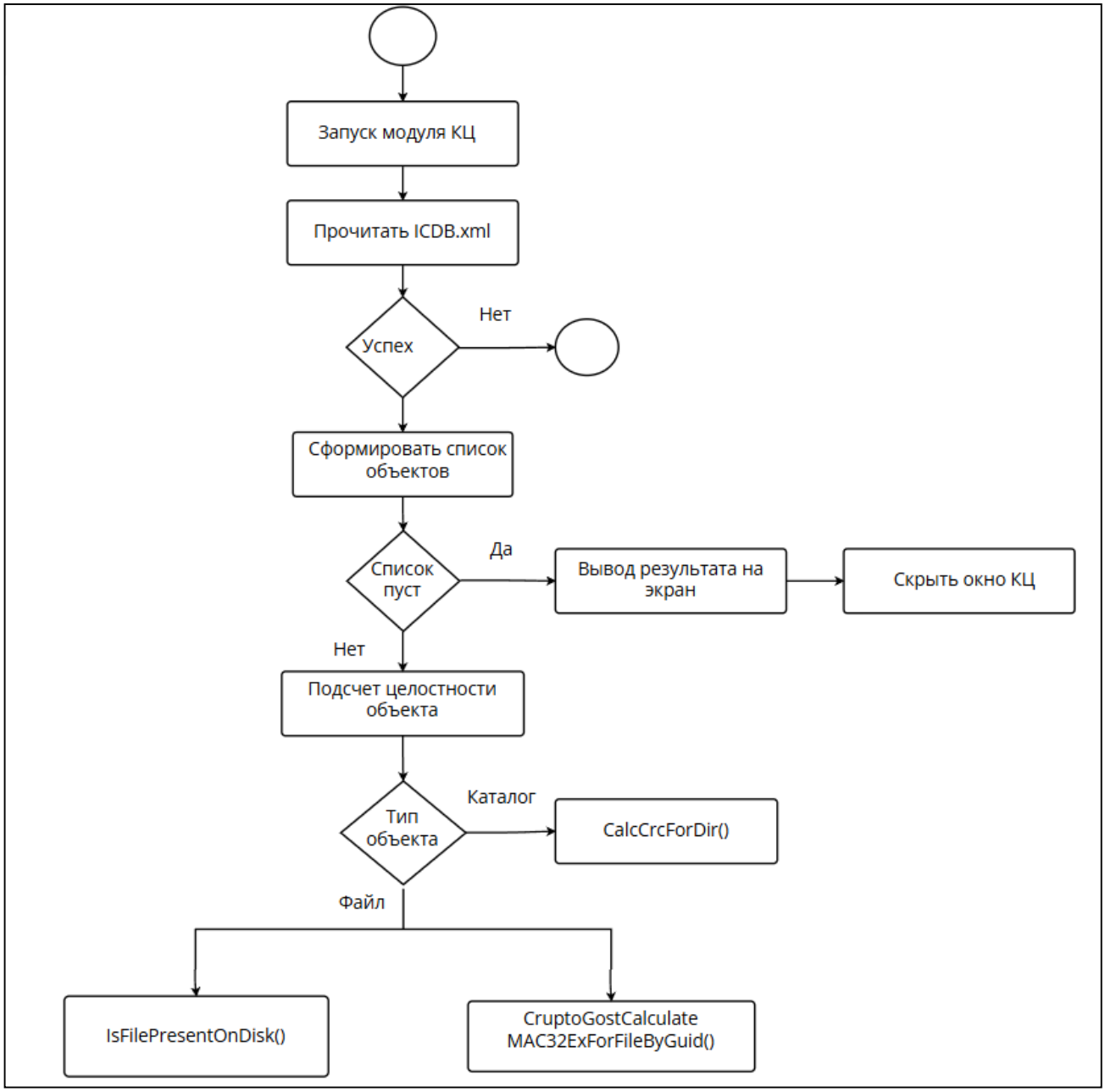


Рисунок 9 – Алгоритм работы функции MakeIntegrityControlDlg()

Эта функция инициирует последовательность действий, направленных на выявление возможных изменений и нарушений целостности файлов или каталогов, входящих в контрольную область. В первую очередь внутри нее

вызывается функция ReadAndParseIcdb(), которая занимается чтением и разбором XML файла, содержащего информацию о контрольных объектах, их параметрах и ожидаемых значениях. После успешного чтения и парсинга данных в дальнейшем формируется единый общий список всех объектов, поставленных на контроль, с помощью функции InitializeListHead(). Эта функция создает и инициализирует корень списка, к которому затем добавляются все файлы, каталоги, а также ключи и параметры, подлежащие мониторингу. Таким образом, создается полная структура данных, охватывающая все элементы для проверки.

После подготовки списка происходит инициализация окна интерфейса, предназначенного для отображения прогресса и результатов проверки целостности, а также элементов управления, которые позволяют пользователю взаимодействовать с процессом и просматривать актуальные данные. После этого запускается функция OneIntegrityControl(), которая реализует цикл проверки: она последовательно перебирает все объекты из подготовленного списка, предварительно распознав тип объекта (каталог или файл), и для каждого из них выполняет проверку на возможную модификацию или нарушение целостности, сравнивая текущие параметры с ожидаемыми или сохраненными значениями.

Для проверки нарушений в каталоге используется функция CalcCrcForDir(), в ходе которой подсчитывается MAC. MAC (Message Authentication Code) — это короткий массив битов, который добавляется к сообщению и предназначен для защиты от несанкционированных изменений. Если вычисленное значение MAC совпадает с полученным, получатель с высокой вероятностью считает, что сообщение дошло без изменений. Пример подсчета представлен ниже:

```
//подсчитаем MAC от имен файлов
UINTNpos = 0;
while (fileList)
{
```

```
        /** Получить имя файла */
        CHAR16* fileName = fileList + pos;
        pos += StrLen(fileName) + 1;
        /** Добавить полный путь */
        CHAR16* fullPath =
(CHAR16*)AllocateZeroPool(FIXED_PATH_LEN * sizeof(CHAR16));
        CopyMem(fullPath, DirInfo.pFileName,
StrLen(DirInfo.pFileName) * sizeof(CHAR16));
        CopyMem(fullPath + StrLen(DirInfo.pFileName), fileName,
StrLen(fileName) * sizeof(CHAR16));
        CalcCRC((UINT32*)fullPath, FIXED_PATH_LEN *
sizeof(CHAR16), &mac);
        FreePool(fullPath);

        if (pos  == fileListSize)
        {
             break;
        }
  }
  FreePool(fileList);

  //подсчитаем MAC от имен каталогов
  pos = 0;
  while (dirList)
  {
        /** Получить имя файла */
        CHAR16* dirName = dirList + pos;
        pos += StrLen(dirName) + 1;
        /** Добавить полный путь */
        CHAR16* fullPath =
(CHAR16*)AllocateZeroPool(FIXED_PATH_LEN * sizeof(CHAR16));
        CopyMem(fullPath, DirInfo.pFileName,
StrLen(DirInfo.pFileName) * sizeof(CHAR16));
        CopyMem(fullPath + StrLen(DirInfo.pFileName), dirName,
StrLen(dirName) * sizeof(CHAR16));
        CalcCRC((UINT32*)fullPath, FIXED_PATH_LEN *
sizeof(CHAR16), &mac);
        FreePool(fullPath);

        if (pos == dirListSize)
        {
             break;
        }
  }
  FreePool(dirList);

  //вернем MAC
  (*CalcMac) = mac;
```

Проверка модификаций в файле происходит в несколько этапов:

– наличие файла и его переименование с помощью функции IsFilePresentOnDisk();

– модификация содержимого с помощью функции CryptoGostCalculateMAC32ExForFileByGuid().

Результаты каждой проверкиотображаются в окне интерфейса, а также сохраняются в файл благодаря функции Save_ICDB_List(). Для того чтобы определить конкретное местоположение и носитель, на который будет сохранен файл с результатами, необходимо предварительно выбрать целевой диск или носитель в специальном окне выбора. Это окно создается и управляется посредством функции MakeSystemDlg(). После того как пользователь кликнет или выберет нужный диск из предоставленного списка, его том (Volume) автоматически будет добавлен в соответствующую узловую структуру конфигурационного файла Config.xml. Этот процесс обеспечивает автоматическую интеграцию выбранного носителя в рабочий процесс программы и позволяет далее, при сохранении файла, использовать данную информацию для правильного определения назначения и пути сохранения.

3.2 Описание работы приложения

После успешной инициализации аппаратной платформы модуля доверенной загрузки (АПМДЗ) система активирует модуль локальной идентификации и аутентификации, который использует многоуровневую систему проверки подлинности пользователя. После прохождения данного этапа блок АПМДЗ инициирует контроль целостности файловой структуры, осуществляется последовательный анализ защищённых объектов. В случае обнаружения нарушения целостности (если хотя бы у одного параметра или файла не совпадет значение контрольной суммы с эталонным значением, хранящимся в защищённой области системы доверенной загрузки) система информирует об этом и активирует механизм защиты. Для обычных

пользователей это означает полную блокировку процесса загрузки с выводом детализированного сообщения о характере нарушения, при этом все данные о событии фиксируются в электронном журнале с привязкой к временным меткам и данным аутентификации. Администраторы системы получают расширенные возможности - после прохождения дополнительной проверки подлинности они могут либо проигнорировать конкретное нарушение (с обязательной записью в журнал аудита), либо внести корректировки и пересчитать целостность. Схема деятельности представлена на рисунке 10.

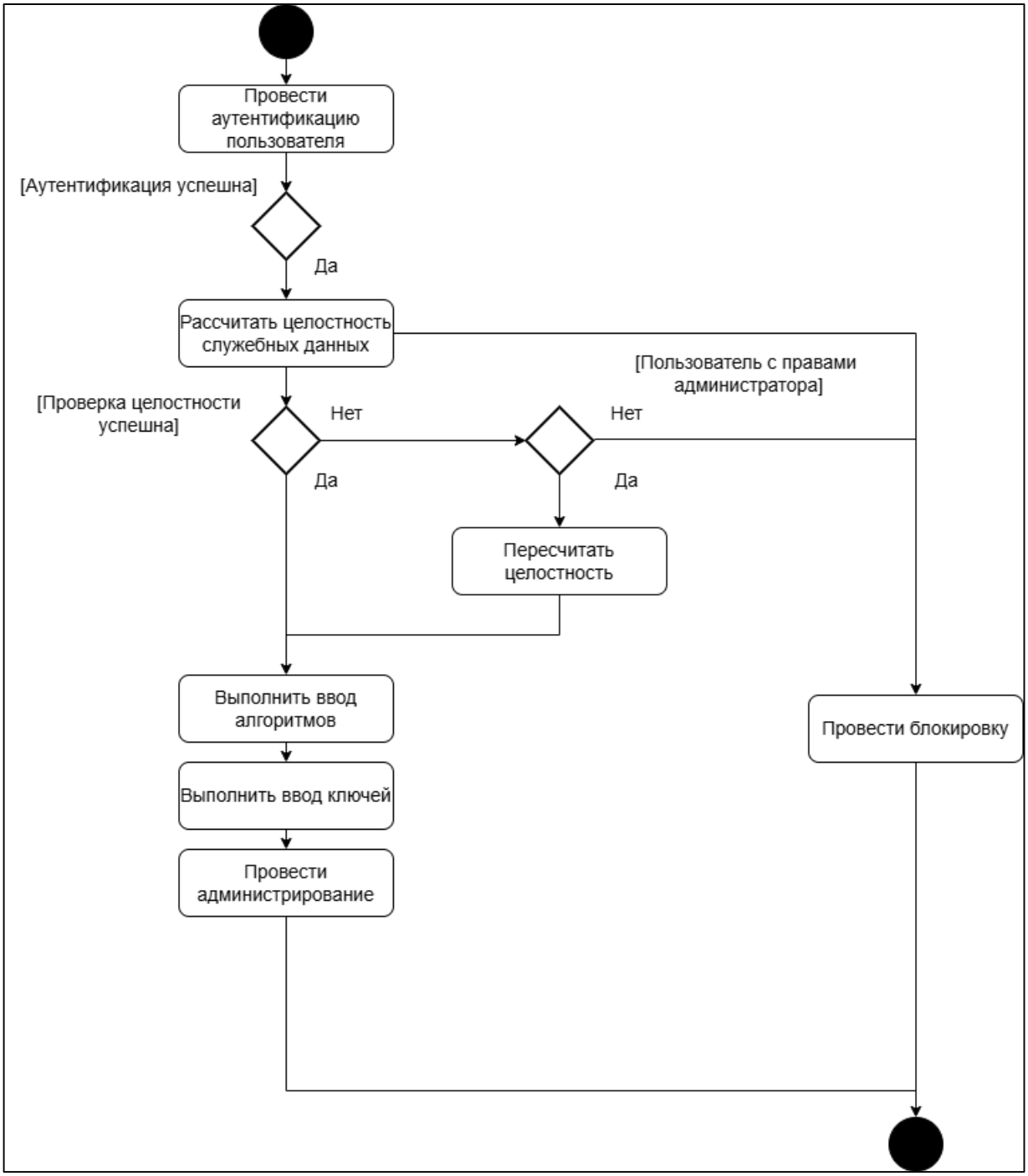


Рисунок 10 – Диаграмма дятельности

3.3 Используемые протоколы при работе с файловыми системами

В контексте разработки для UEFI важным аспектом является работа с файлами и файловой системой, которая реализована с помощью различных протоколов и структур.

EFI_FILE_PROTOCOL является базовым интерфейсом для работы с файлами в UEFI-среде. Он предоставляет методы для открытия, чтения, записи, закрытия файлов, а также управления их атрибутами и положением. Этот протокол используется для доступа к файлам на различных носителях, что особенно важно при реализации функций контроля целостности, когда необходимо проверять содержимое файлов или сохранять результаты проверок.

EFI_FILE_SYSTEM_INFO предоставляет базовую информацию о файловой системе, такую как ее размер, свободное пространство и метку тома. Этот протокол используется для мониторинга состояния файловых систем, что особенно важно при работе с модулем контроля целостности.

EFI_FILE_INFO содержит метаданные о файле, такие как его размер, атрибуты, время создания и изменения. Эта структура используется для получения информации о файлах в процессе проверки их целостности. При сравнении текущего состояния файла с эталонным значением модуль может анализировать изменения в размере или времени модификации, что может свидетельствовать о потенциальных угрозах.

FULL_VOLUME_INFO представляет собой расширенную информацию о томе или разделе файловой системы. В отличие от стандартных структур UEFI, она может включать дополнительные сведения, такие как тип файловой системы, параметры форматирования, метаданные и списки файлов.

Взаимодействие этих протоколов и структур данных позволяет реализовать комплексный контроль целостности файлов и каталогов в UEFI-среде.

3.4 Руководство пользователя

В процессе использования программного модуля пользователь или администратор может выполнять различные действия, направленные на частности, он может:

– посмотреть объекты, которые уже стоят на контроле;

– выбрать файлы/каталоги для проверки;

– указать том, на котором будет храниться результат подсчета целостности.

После запуска программы пользователь видит окно интерфейса, представленного на рисунке 11, которое, которое разделено на несколько частей. Детально разберем каждую из этих частей, чтобы понять их назначение и работу.

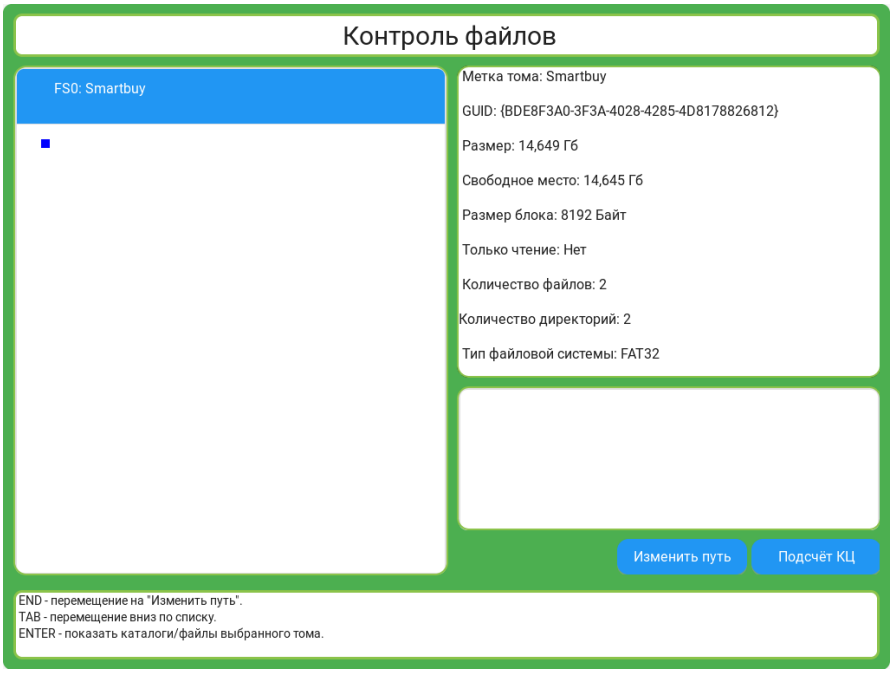


Рисунок 11 – Пользовательское окно программного модуля

В левом разделе расположена панель, отвечающая за отображение файловых систем и их содержимого. Здесь представлены все подключенные тома, каталоги и файлы, которые пользователь может выбрать для дальнейших действий. Для удобства ориентации и быстрого доступа к нужным объектам используются специальные виджеты — кнопки и индикаторы, с помощью которых сразу можно определить, на каком именно объекте или уровне системы находится пользователь. Эти визуальные элементы выполнены так, чтобы обеспечить максимально интуитивное понимание, облегчая навигацию по дереву файлов и структур данных, на рисунке 12 изображен список с наглядным разделением файлов и каталогов, примеры возможных символов для подобного разграничения представлены на рисунке 13.

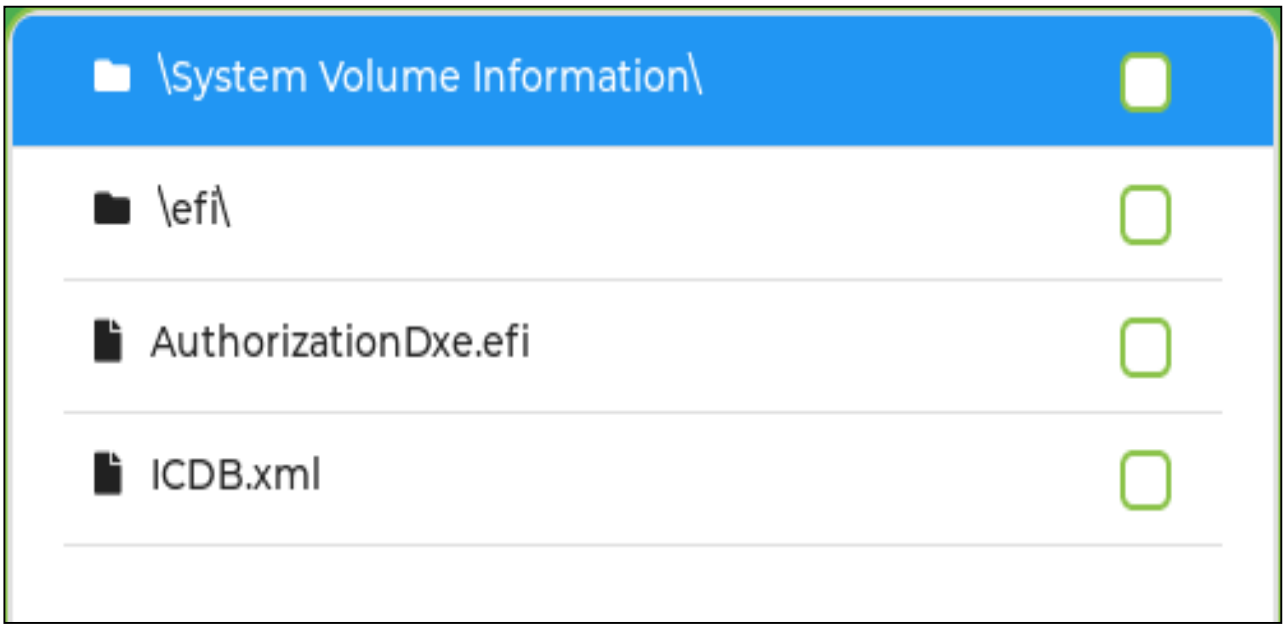


Рисунок 12 – Список каталогов и файлов

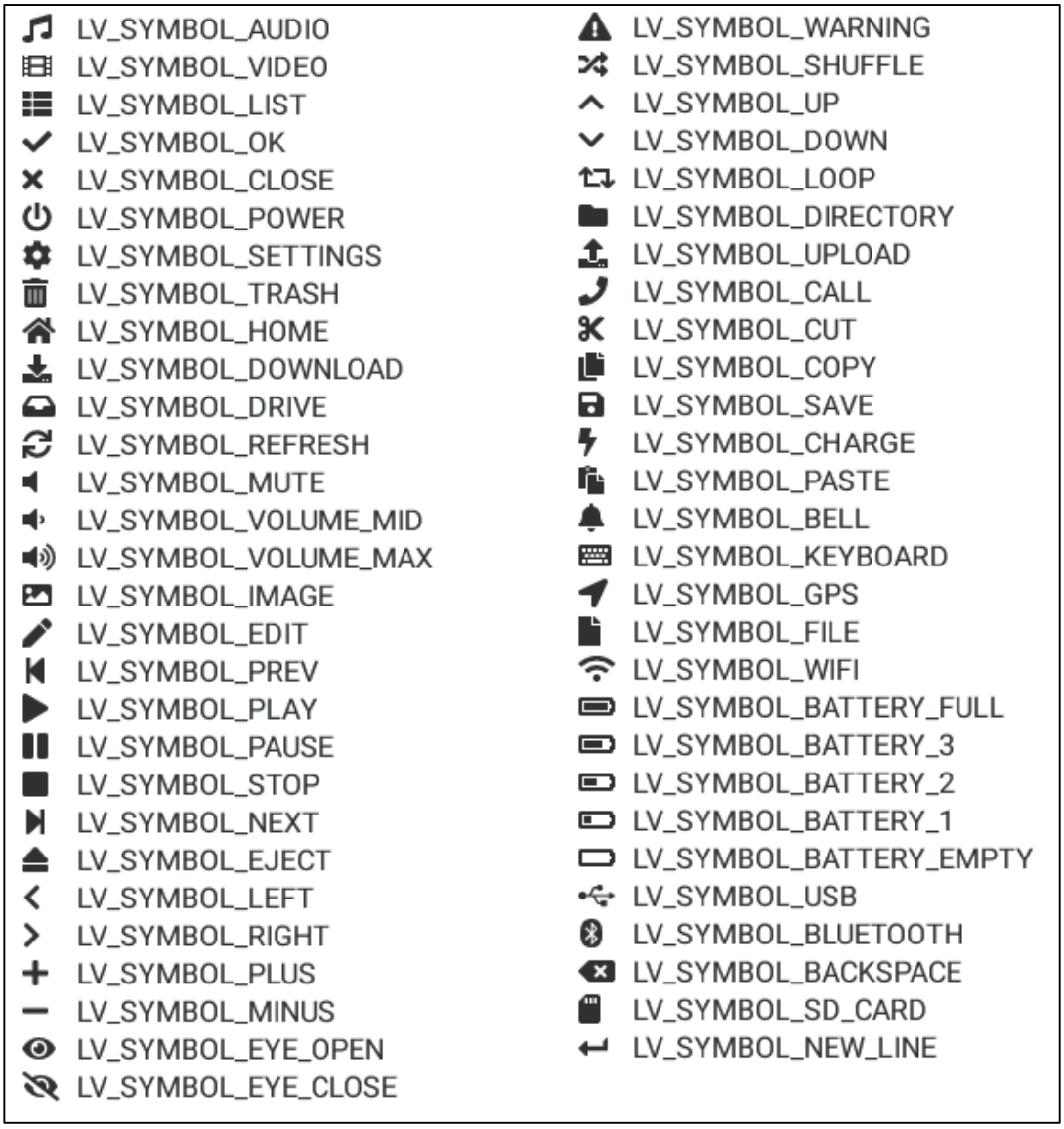


Рисунок 13 – Виджеты *LVGL* библиотеки

Пользователь имеет возможность легко и удобно заходить в любую из выбранных областей или разделов системы благодаря простому клику мыши или же при помощи клавиши ENTER, а также возвращаться на уровень выше, используя клавишу BACKSPACE. Для того чтобы всегда было понятно, какие клавиши и для каких действий предназначены, в нижней части интерфейса представлены подсказки и напоминания, рисунок 14.

Рисунок 14 – Навигационные подсказки

Для того чтобы пользователь мог быстро и наглядно увидеть, какие именно файлы уже находятся под контролем системы, а какие еще нет - на каждой соответствующей кнопке списка предусмотрен чекбокс. Этот чекбокс служит визуальным индикатором выбранных объектов: активное состояние данного элемента сигнализирует о том, что данный файл или каталог уже контролируются, что изображено на рисунке 15, а отключенное – возможности провести над ними проверку.

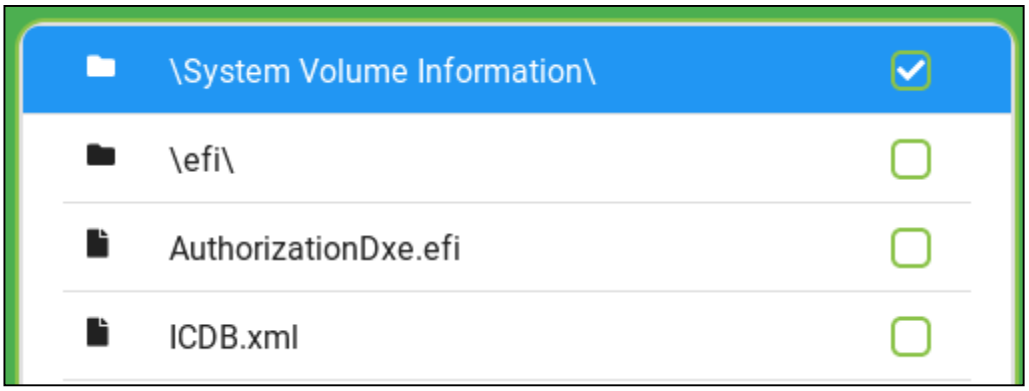


Рисунок 15 – Каталог поставлен на контроль

После того, как пользователь нажимает на выбранный объект, этот объект отображается в правом нижнем окне интерфейса, которое специально предназначено для хранения и отображения списка файлов или каталогов, выбранных для проведения процедуры подсчета их целостности (рисунок 16).

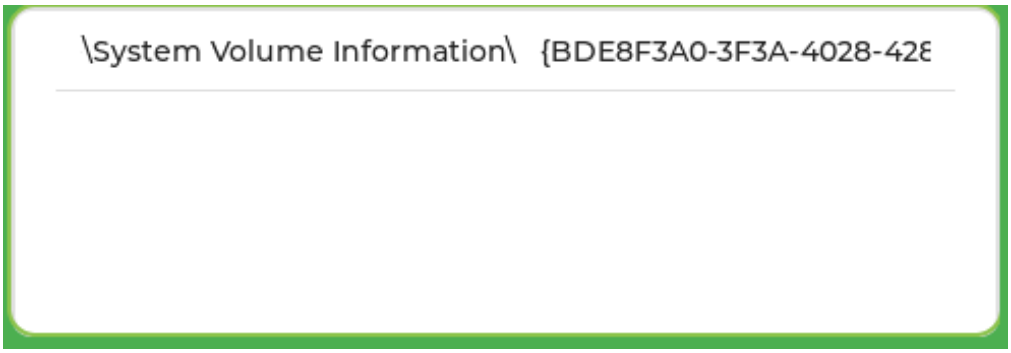


Рисунок 16 – Список объектов, поставленных на контроль

После того, как пользователь завершает добавление необходимых файлов и каталогов в список, он переходит к следующему шагу — нажатию на кнопку «Подсчет КЦ». В этот момент программа инициирует процесс проверки целостности выбранных объектов. В процессе выполнения операции пользователь может следить за прогрессом или статусом проверки, а по завершении программа выводит конечный результат на экран. В случае обнаружения модификации объекта – программа информирует об этом пользователя, выделяя объект красным цветом и указывая на ошибку в соответствующей строке, пример представлен на рисунке 17.

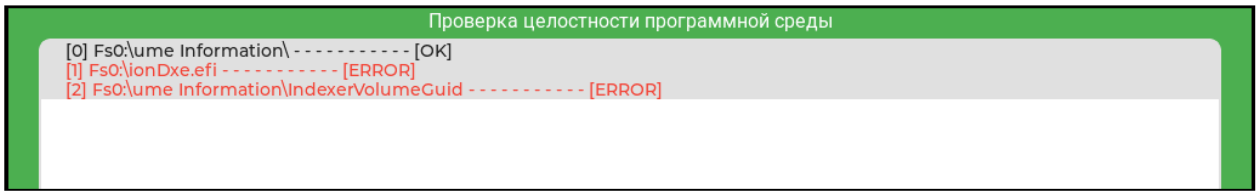


Рисунок 17 – Результат проверки целостности

После вывода результата на экран он будет сохранен в XML файле. Для выбора тома, на котором будет храниться этот XML файл, предусмотрена специальная кнопка "Изменить путь". При нажатии на кнопку открывается окно, в котором пользователь может выбрать системный том из списка доступных дисковых томов, изображение окна представлено на рисунке 18.

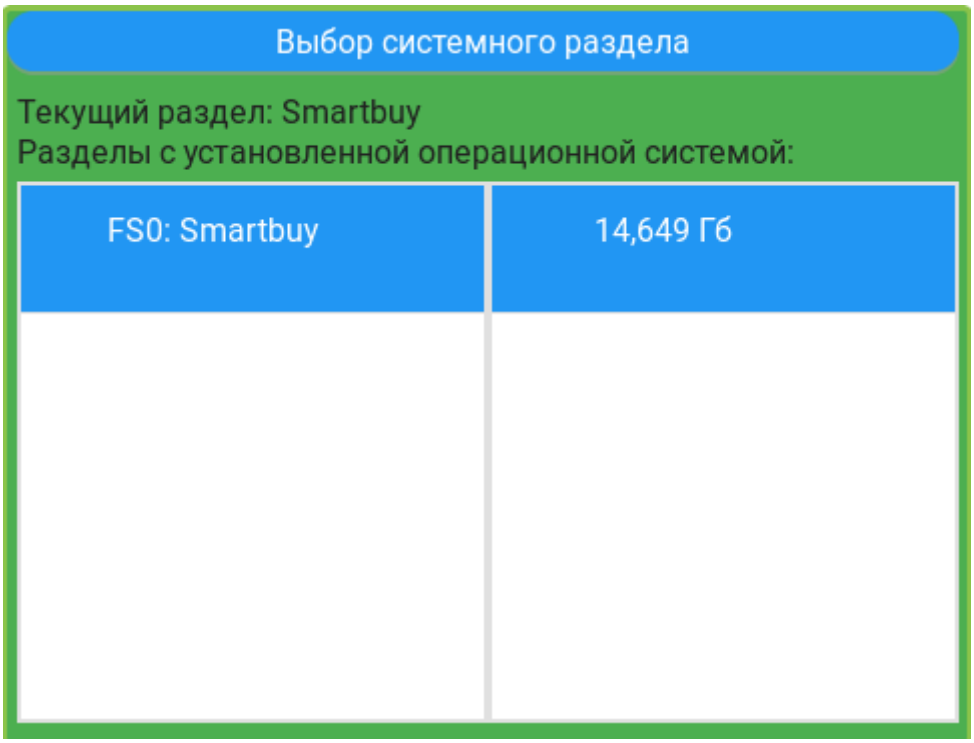


Рисунок 18 – Выбор системного раздела

Как только пользователь выбирает какой-либо том из этого списка, появляется окно сохранения изменений (рисунок 19) и после подтверждения, нажав на кнопку «Да», том автоматически становится системным по умолчанию.

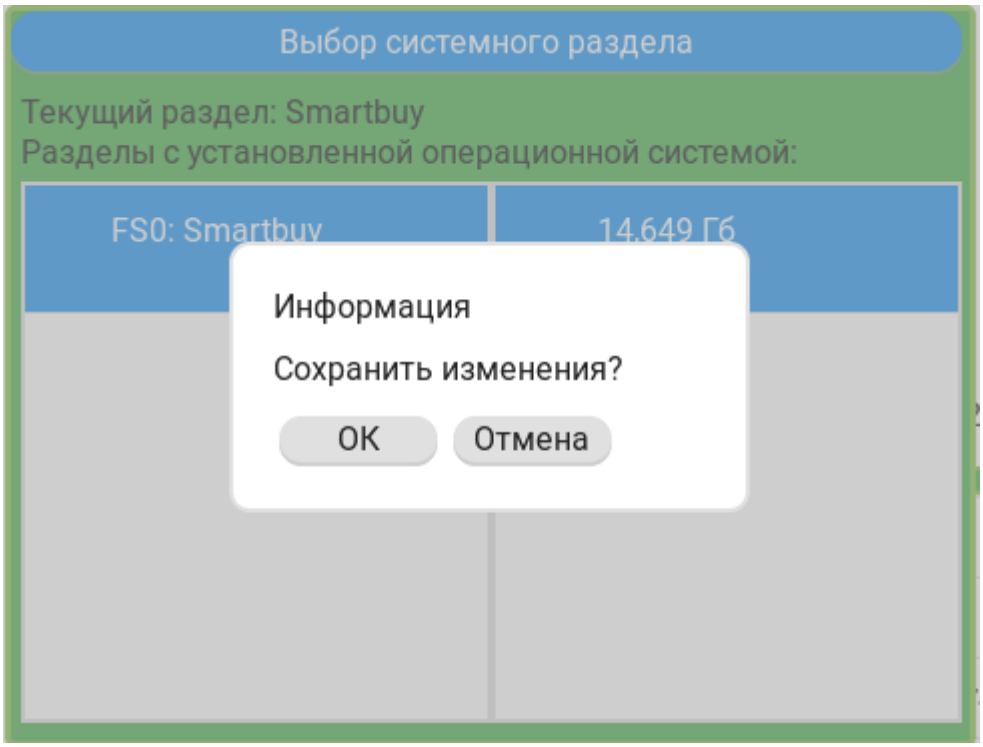


Рисунок 19 – Сохранение изменений после выбора системного тома

После выбора системного тома и подтверждения выбора, в XML файле, содержащем настройки программы, автоматически подставляется соответствующее значение Guid выбранного тома. Таким образом, программа сохраняет информацию о томе, на котором будут храниться настройки и результат работы, что обеспечивает удобство и автоматизацию процесса настройки и хранения данных.

3.5 Тестирование приложения

При разработке программного модуля одной из важных задач является его тестирование. Проведение тестирования позволяет не только выявить возможные ошибки в коде, но и обнаружить проблемные моменты в функциональности, а также убедиться в том, что все предварительно поставленные задачи и требования выполняются в соответствии с ожидаемым результатом. В рамках текущего тестирования будут проверены и протестированы следующие функции приложения:

– изменение названия каталога;
– изменение названия файла;
– удаление каталога;
– удаление файла;
– изменение содержимого файла.

3.5.1 Изменение названия каталога

Таблица 5 – Тест-кейс №1 (изменение названия каталога)

Тест-кейс №1 предназначен для проверки нарушения целостности при попытке модифицировать название каталога		
Шаги	Данные	Ожидаемый результат
Создать несколько папок на файловой системе и поставить их на контроль	«book 1», «book 2», «book 3»	Предварительный подсчет прошел успешно
Модифицировать название двух папок и повторно подсчитать целостность	«!book 1», «!book 3»	Нарушение целостности, вывод объекта с ошибкой

Для проверки корректности работы данного механизма создадим на нулевом томе три папки с названиями «book 1», «book 2» и «book 3» (рисунок 20).

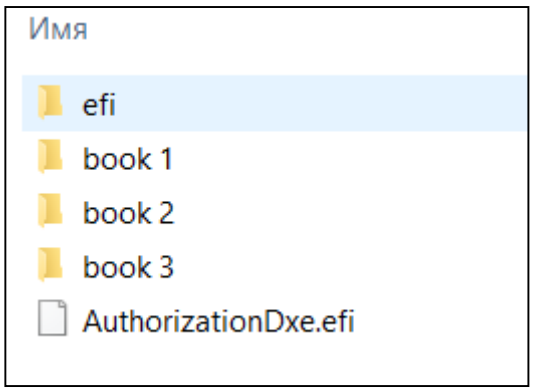


Рисунок 20 – Создание папок на нулевом томе

Поставим созданные папки на контроль (рисунок 21).

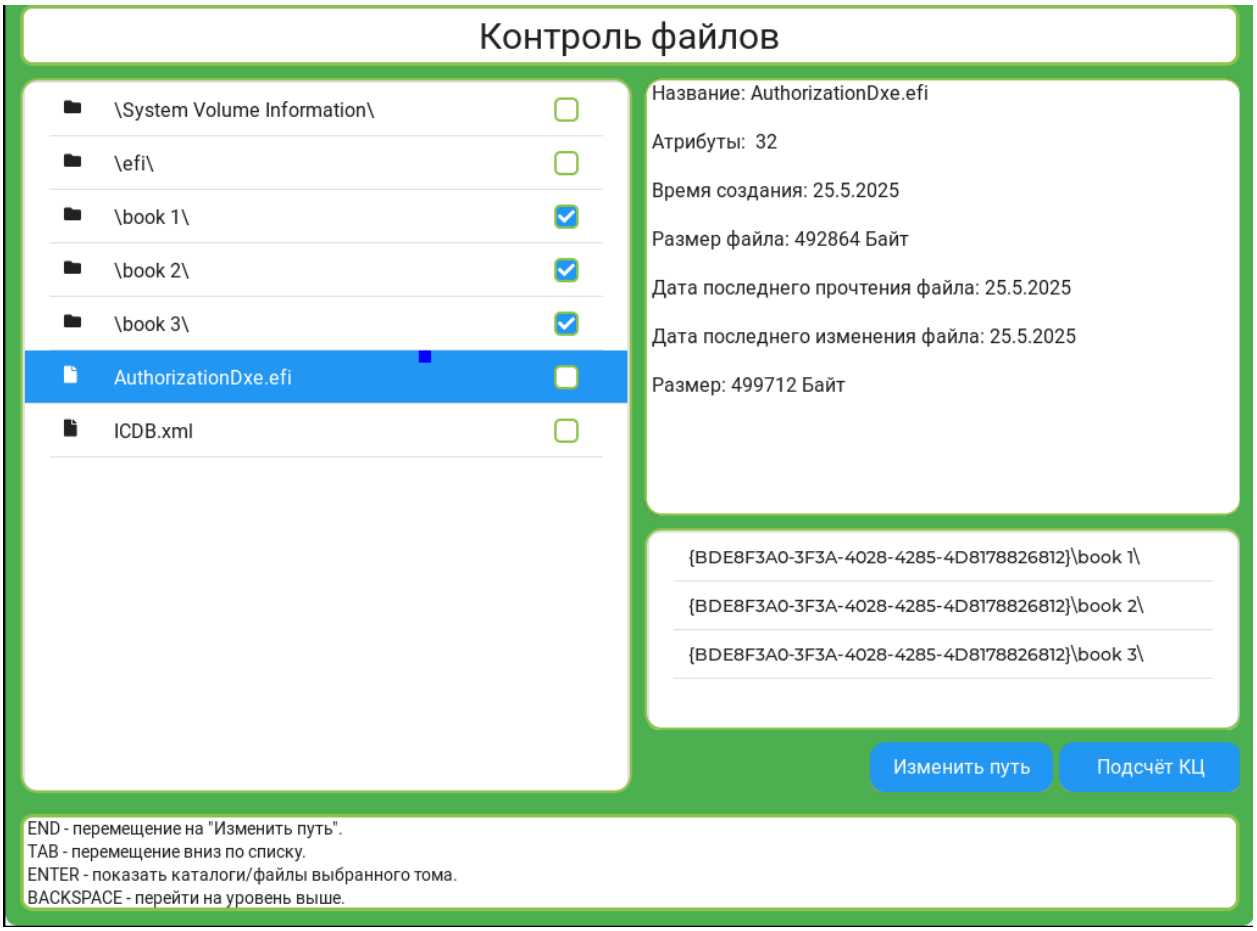


Рисунок 21 – Постановка каталогов на контроль

Первичный анализ целостности каталогов до внесения каких-либо изменений показан на рисунке 22.

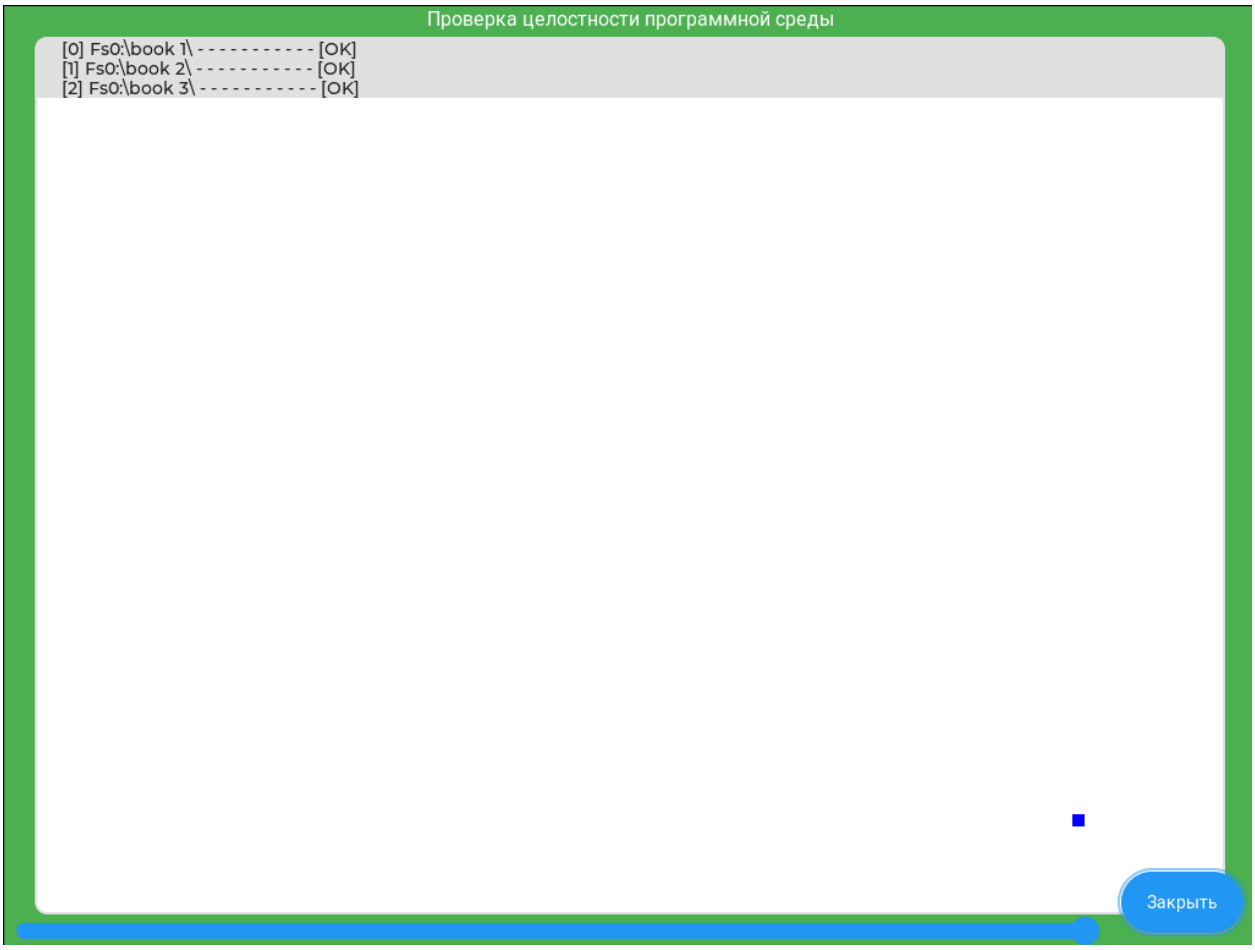


Рисунок 22 – Целостность каталогов до переименования

Далее для проверки реакции системы изменим названия папок «book 1» на «!book 1» и «book 3» на «!book 3» (рисунок 19). После внесения этих изменений проведем повторный подсчет целостности данных, который представлен на рисунке 23.

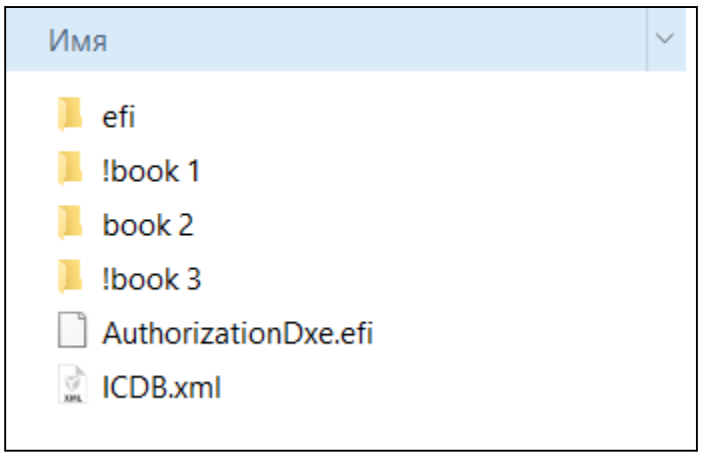


Рисунок 23 – Переименование каталогов

Результат повторного подсчета целостности представлен на рисунке 24. Объекты, название которых было изменено, обозначены с ошибкой, что подтверждает корректность работы системы оповещения о нарушениях.

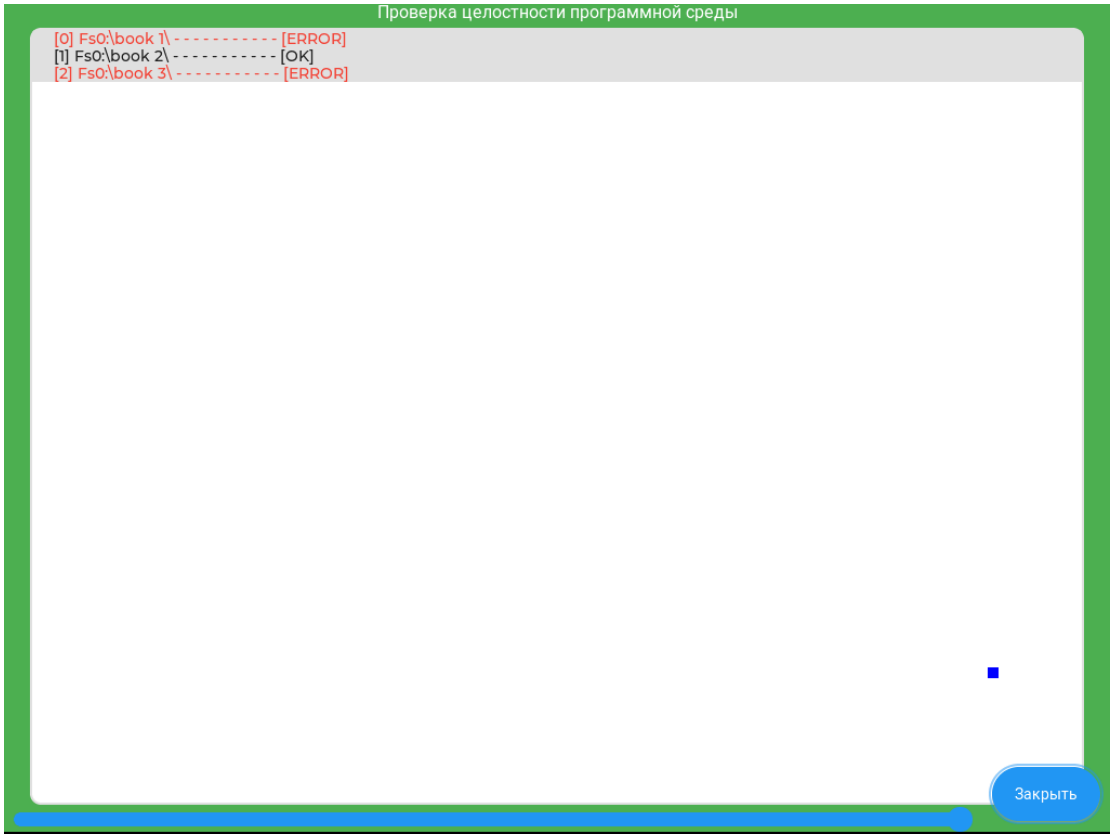


Рисунок 24 – Целостность каталогов после переименования

3.5.2 Изменение названия файла

Таблица 6 – тест-кейс №2 (изменение названия файла)

Тест-кейс №2 предназначен для проверки нарушения целостности при попытке модифицировать название файла		
Шаги	Данные	Ожидаемый результат
Создать несколько файлов в каталоге	«file 1», «file 2», «file 3»	Предварительный подсчет прошел успешно
Модифицировать название двух папок и повторно подсчитать целостность	«!file 1», «file 333»	Нарушение целостности, вывод объекта с ошибкой

Для проверки корректности работы данного механизма в папке «book1» три файла с названиями «file 1», «file 2» и «file 3» (рисунок 25).

Рисунок 25 – Создание файлов в папке book 1

Поставим созданные файлы на контроль. Результат первого подсчета представлен на рисунке 26.

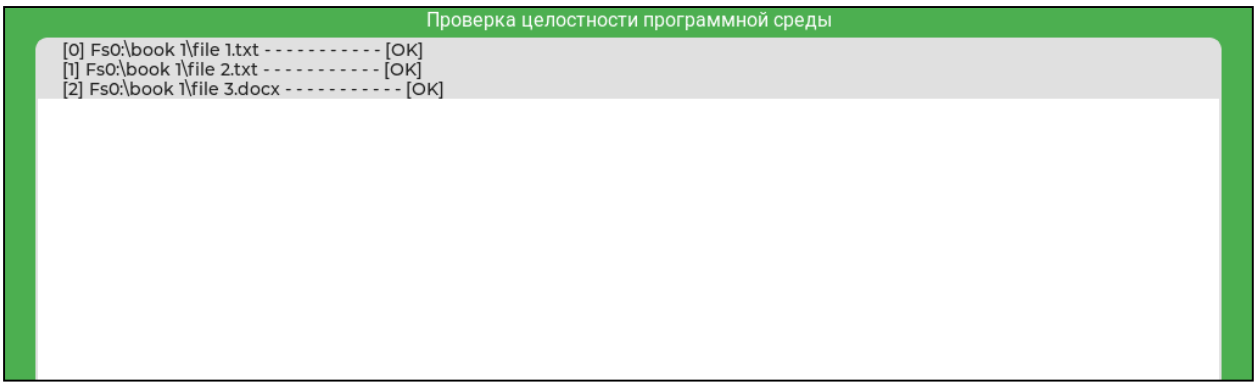


Рисунок 26 – целостность файлов до переименования

Для проверки изменим названия файлов «file 1» на «!file 1» и «file 3» на «file 333» (рисунок 27).

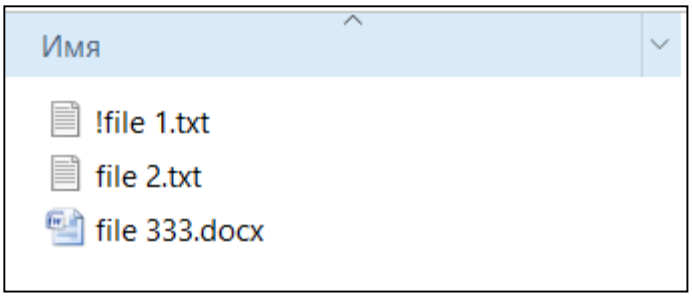


Рисунок 27 – Переименование файлов

Объекты, для которых были внесены изменения в название, обозначены с ошибкой, что свидетельствует о правильности функционирования системы уведомлений о нарушениях (рисунок 28).

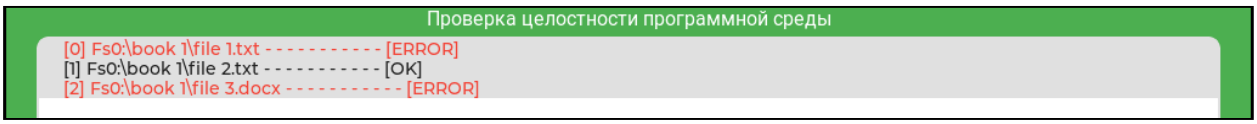


Рисунок 28 – Целостность файлов после переименования

3.5.3 Удаление каталога из списка

Таблица 7 – тест-кейс №3 (удаление каталога из списка)

Тест-кейс №3 предназначен для проверки нарушения целостности при попытке удалить директорию		
Шаги	Данные	Ожидаемый результат
Создать несколько папок на файловой системе и поставить их на контроль	«book 1», «book 2», «book 3»	Предварительный подсчет прошел успешно
Удалить из списка каталог и повторно подсчитать целостность	Объект на удаление - «book 2»	Нарушение целостности, вывод объекта с ошибкой

В процессе работы программа будет проверять наличие данного каталога в системе и отслеживать любые изменения его состояния. В случае, если по каким-либо причинам объект был удален или исчез из системы, программа должна своевременно обнаружить данное нарушение и уведомить об этом.

Обратимся к уже созданным папкам с названиями «book 1», «book 2» и «book 3» (рисунок 29) и поставим их на контроль (рисунок 30).

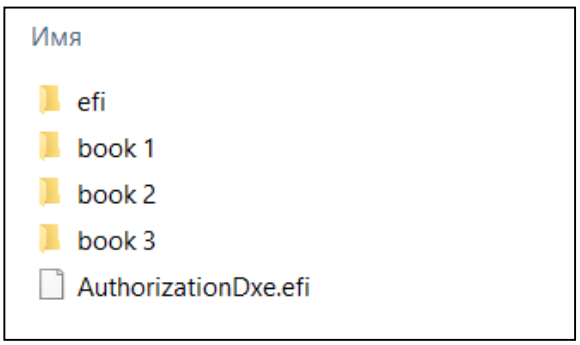


Рисунок 29 – Список каталогов до удаления

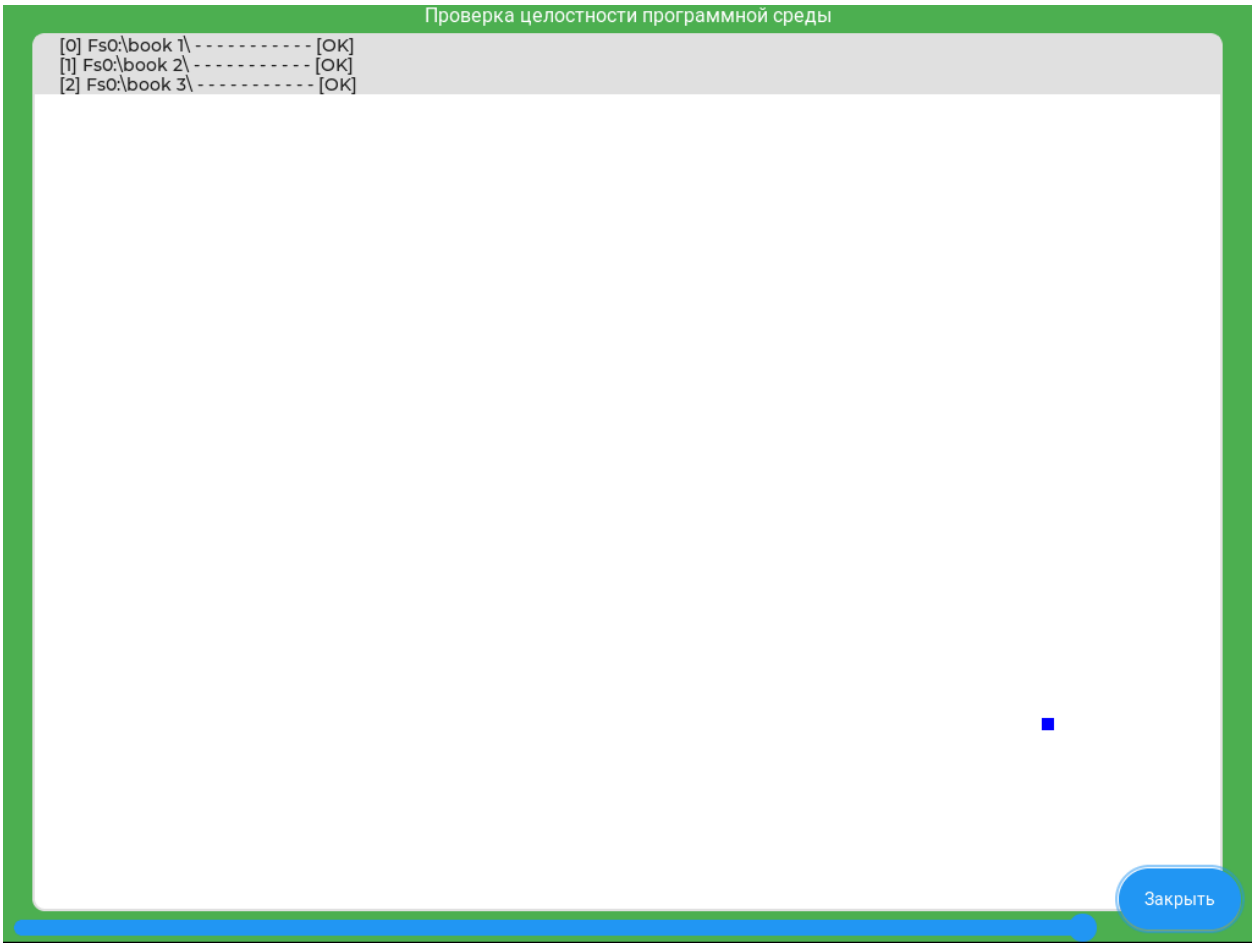


Рисунок 30 – Целостность каталогов до удаления

Удалим из списка объект с именем «book 2» (рисунок 31) и проведем повторную проверку. Удаленный объект указан в открывшемся окне с ошибкой, а значит - данная функция отрабатывает корректно. Результат представлен на рисунке 32.

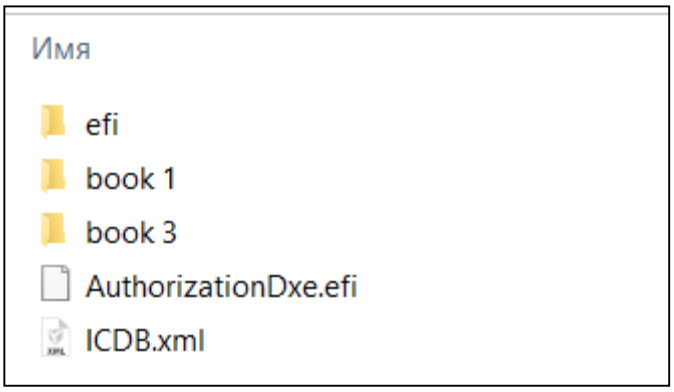


Рисунок 31 – Список каталогов после удаления

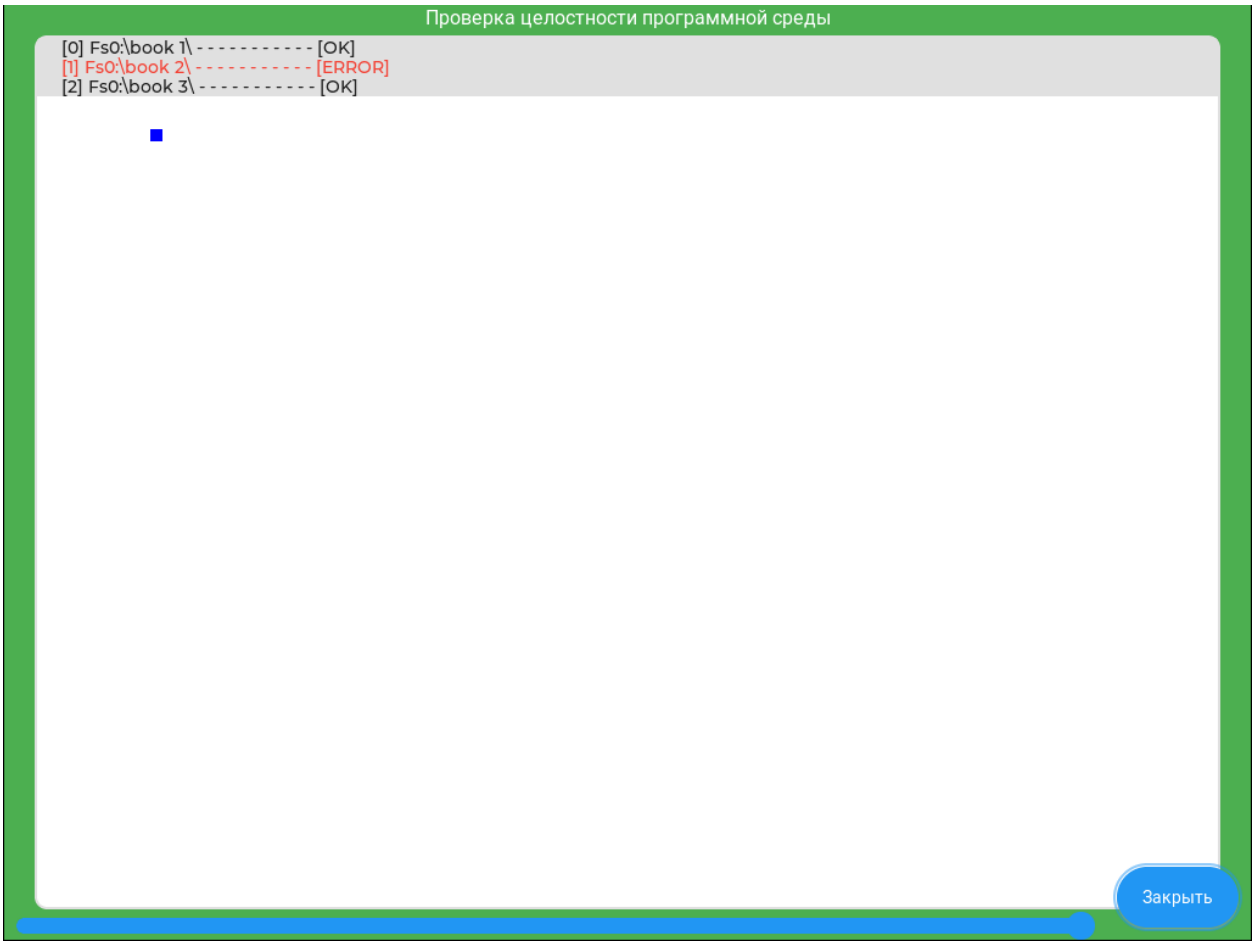


Рисунок 32 – Целостность каталогов после удаления

3.5.4 Удаление каталога из списка

Таблица 8 – тест-кейс №4 (удаление файла из списка)

Тест-кейс №4 предназначен для проверки нарушения целостности при попытке удалить файл		
Шаги	Данные	Ожидаемый результат
Создать несколько файлов в каталоге и поставить их на контроль	«!file1», «file2» «file 333»	Предварительный подсчет прошел успешно
Удалить из списка файл и повторно подсчитать целостность	Объект на удаление - «file333»	Нарушение целостности, вывод объекта с ошибкой

По аналогии с каталогами, проведем проверку над удалением файла из списка. Обратимся к уже созданным папкам с названиями «!file 1», «file 2» и «file 333» (рисунок 33) и поставим их на контроль (рисунок 34). Результат предварительного подсчета представлен на рисунке 35.

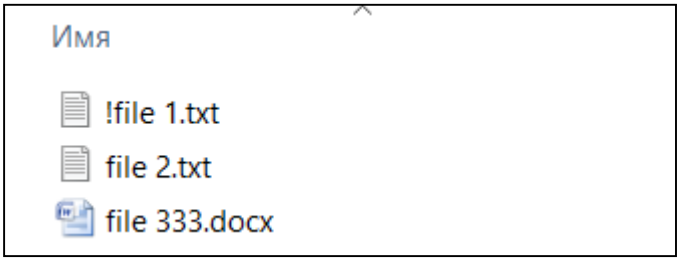


Рисунок 33 – Список файлов до удаления

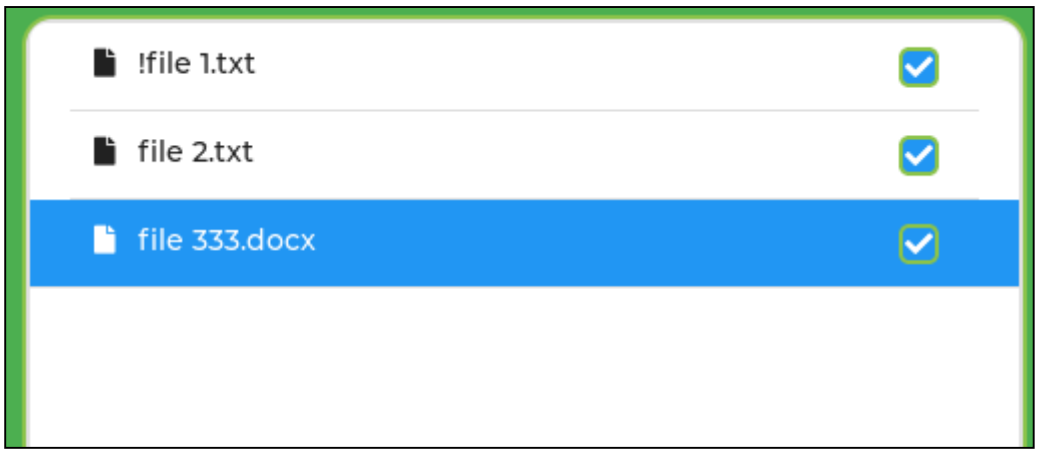


Рисунок 34 – Постановка файлов до удаления на контроль

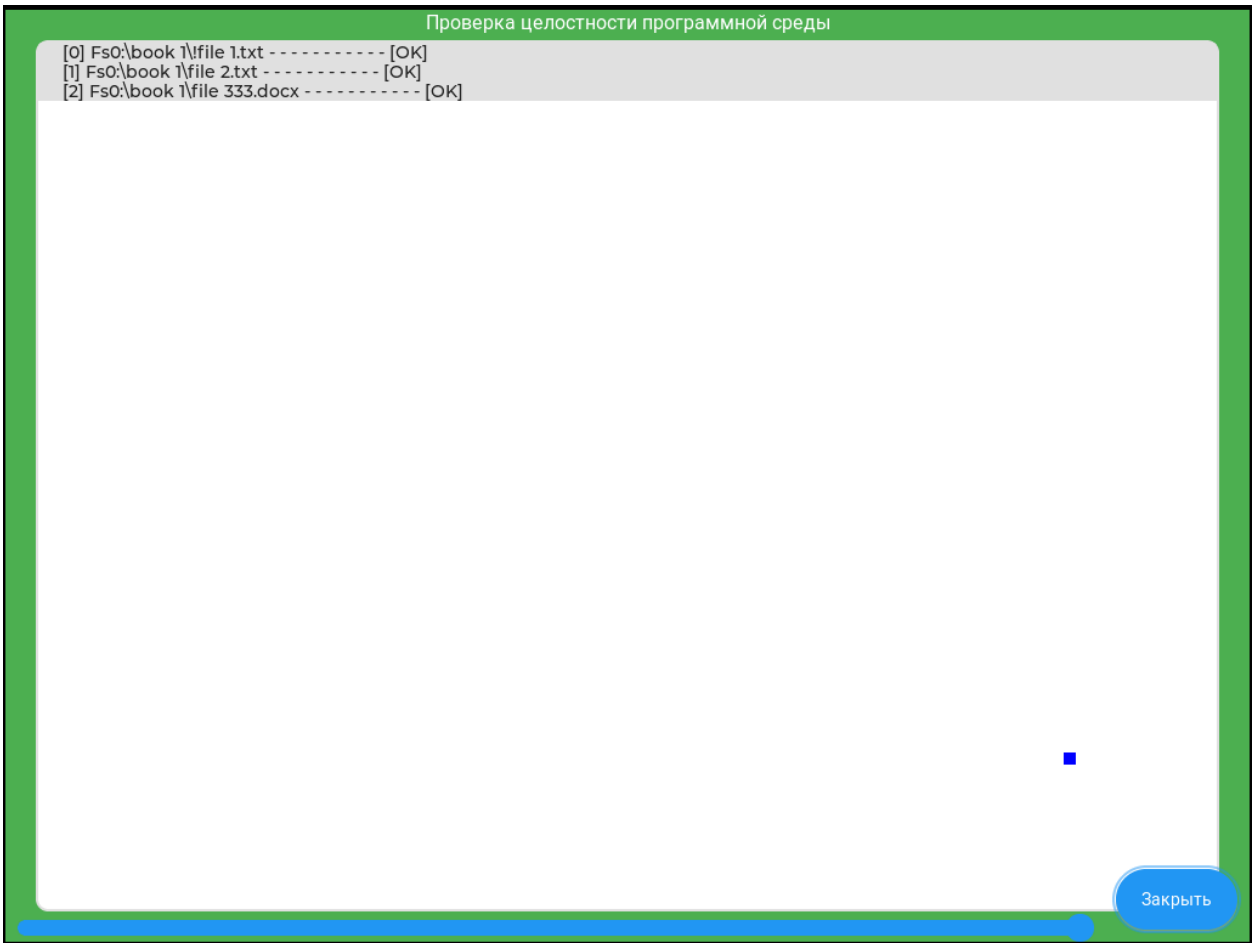


Рисунок 35 – Целостность списка до удаления

Исключим файл с названием «file 333» из списка (рисунок 36) и произведем повторный подсчет, результат которого представлен на рисунке 37.

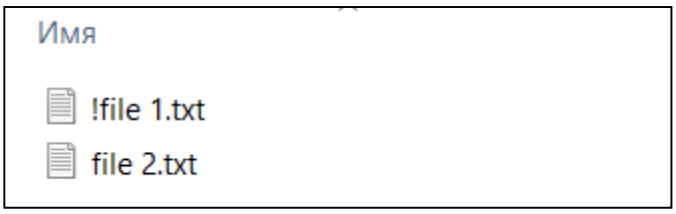


Рисунок 36 – Список файлов после удаления

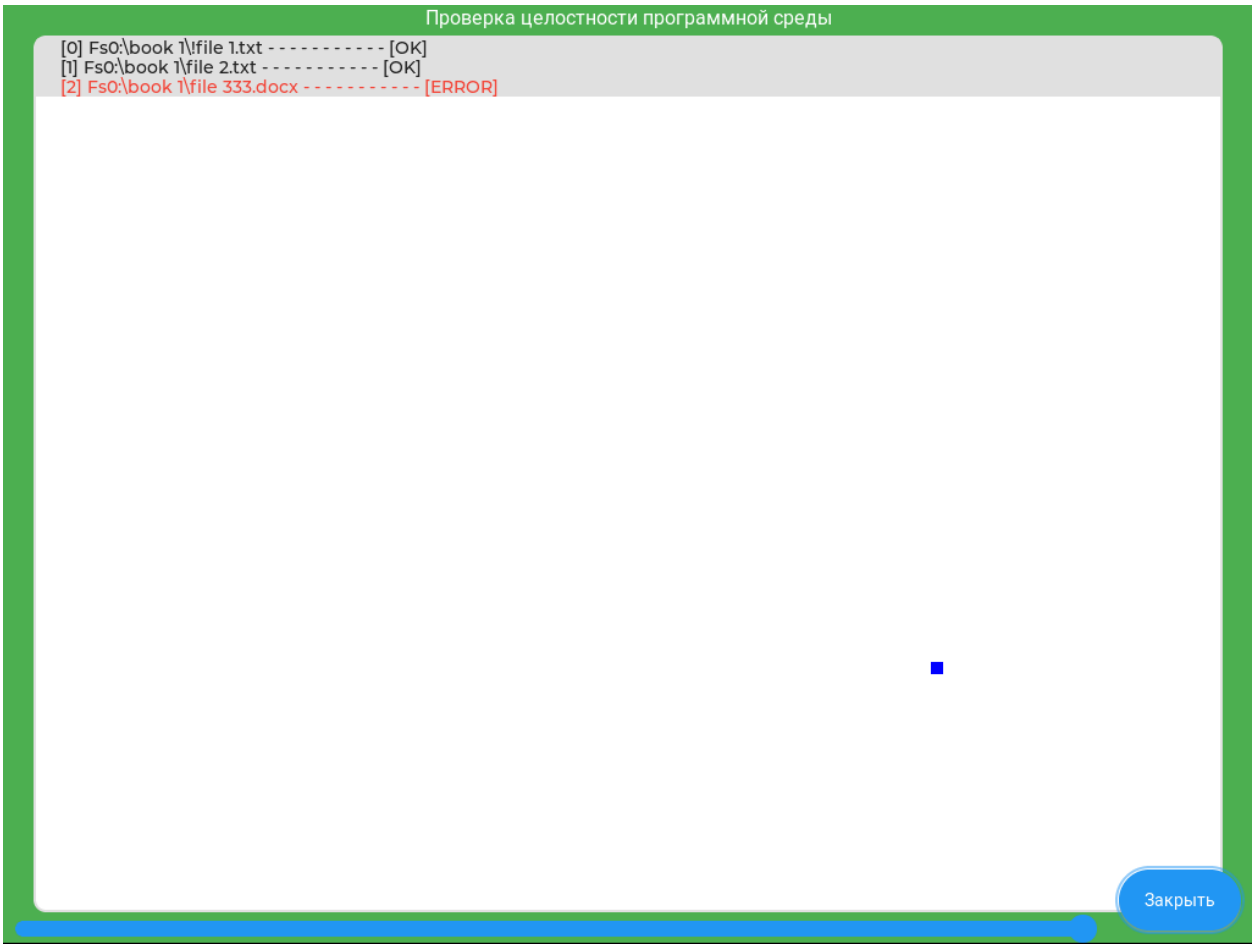


Рисунок 37 – Целостность списка после удаления

Целостность была нарушена, что свидетельствует обуспешнойотработке функции.

3.5.5 Удаление содержимого файла

Таблица 9 – тест-кейс №5 (изменение содержимого файла)

Тест-кейс №4 предназначен для проверки нарушения целостности при попытке модифицировать содержимое файла		
Шаги	Данные	Ожидаемый результат
Добавить в файл текст	«Белеет парус одинокий в тумане моря голубом»	Предварительный подсчет прошел успешно
Заменить текст файла	«У Лукоморья дуб зеленый»	Нарушение целостности, вывод объекта с ошибкой

Программа должна осуществлять отслеживание содержимого файла, который был поставлен на контроль. В случае, если произойдут любые изменения — будь то добавление, удаление или изменение данных внутри файла — программа должна немедленно выявить это нарушение.

На рисунке 38 представлено исходное содержимое файла. После поставки данного файла на контроль (рисунок 39) – программа вывела успешный результат проверки (рисунок 40).

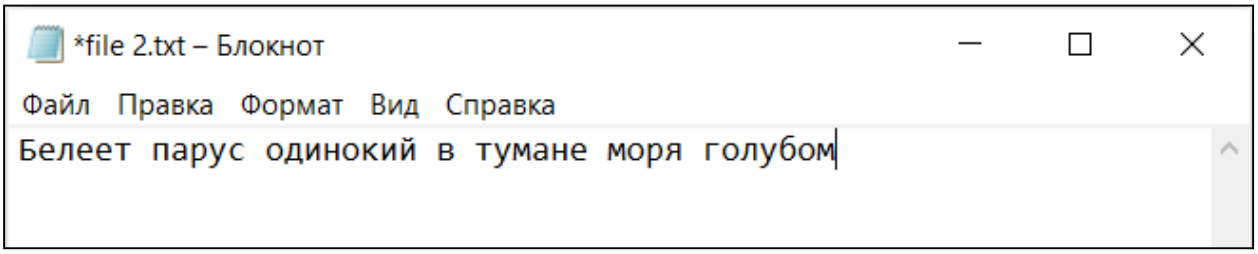


Рисунок 38 – Исходное содержимое файла

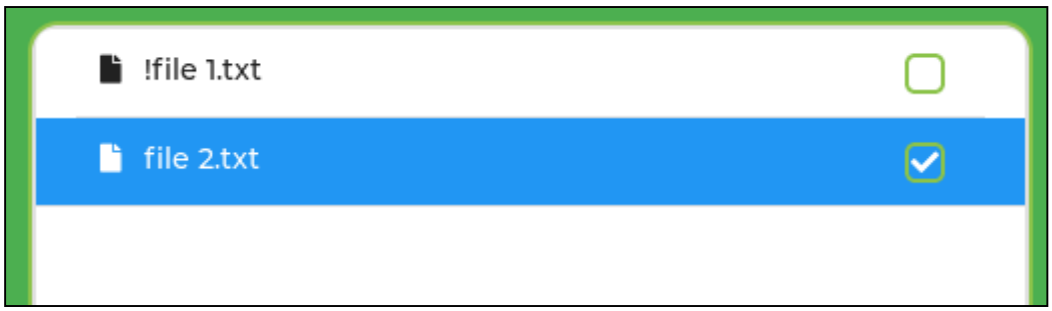


Рисунок 39 – Файл поставлен на контроль

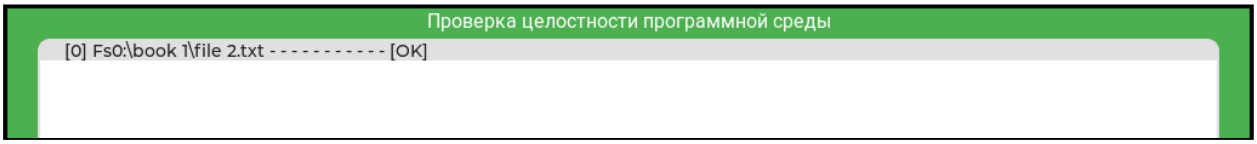


Рисунок 40 – Результат подсчета до изменения содержимого файла

Заменим текст внутри файла (рисунок 41). После внесения необходимых изменений внутри файла, мы повторно запустим процедуру подсчета его целостности и убедимся в том, что программа успешно выявила факт внесения изменений и правильно среагировала на него (рисунок 42).

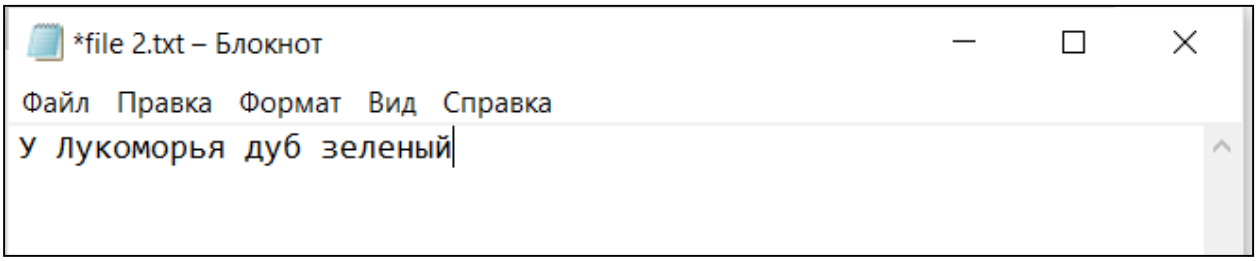


Рисунок 41 – Измененное содержимое файла

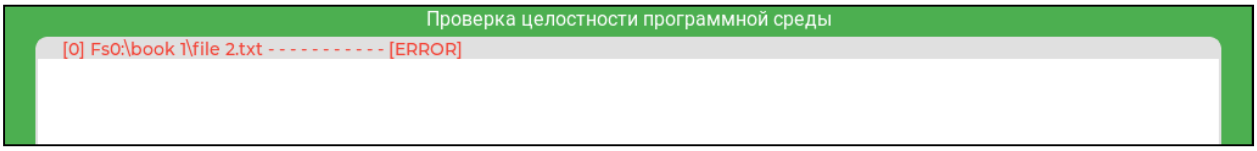


Рисунок 42 – Результат подсчета после изменения содержимого файла

4 Безопасность жизнедеятельности

Использование современных технологических устройств и программного обеспечения связано не только с удобством, но и с определёнными рисками, которые требуют особого внимания. В целях защиты пользователей и разработчиков созданы специальные инструкции и нормативные документы, регулирующие порядок безопасной работы с цифровыми системами. Нарушение этих требований может привести к сбоям в работе оборудования, утечкам конфиденциальной информации или ухудшению физического и психологического состояния пользователя.

4.1 Правила и техника безопасности при работе с ПК

Данная инструкция по охране труда при работе на персональном компьютере (ПК) устанавливает требования охраны труда перед началом, во время и по окончании работы пользователя компьютера (ноутбука), требования охраны труда в аварийных ситуациях.

В целях соблюдения требований охраны труда при работе на персональном компьютере необходимо:

– знать и соблюдать требования электро- и пожаробезопасности, охраны труда и производственной санитарии при выполнении работ на персональном компьютере;

– соблюдать правила личной гигиены;

– знать способы рациональной организации рабочего места;

– иметь четкое представление об опасных и вредных факторах, связанных с выполнением работ на компьютере, знать основные способы защиты от их воздействия;

– пользоваться персональным компьютером (ноутбуком) согласно инструкции по эксплуатации производителя;

– знать порядок действий при поломке, возгорании компьютера, сигналы оповещения о пожаре;
– уметь пользоваться первичными средствами пожаротушения;
– знать месторасположение аптечки и уметь оказывать первую помощь пострадавшему;
– соблюдать установленные режимы труда и отдыха.

4.2 Предварительная подготовка

Подготовить рабочее место:

– отрегулировать освещение на рабочем месте, убедиться в отсутствии бликов на экране;

– проверить правильность подключения оборудования к электросети;

– проверить исправность проводов питания и отсутствие оголенных участков проводов;

– убедиться в наличии заземления системного блока, монитора и защитного экрана;

– протереть антистатической салфеткой поверхность экрана монитора и защитного экрана;

– проверить правильность установки стола, стула, подставки для ног, пюпитра, угла наклона экрана, положение клавиатуры, положение «мыши» на специальном коврике, при необходимости произвести регулировку рабочего стола и кресла, а также расположение элементов компьютера в соответствии с требованиями эргономики и в целях исключения неудобных поз и длительных напряжений тела.

4.3 Завершение работы с ПК

Для этого нужно:

– закрыть все активные программы и задачи;

– извлечь все накопители (диски, флешки);

– отключить компьютер, а затем питание системника;

– выключить вспомогательную технику, бесперебойник, стабилизатор;

– вынуть все кабели из розетки (обесточить технику).

Важно оставить рабочее место чистым, без мусора и посторонних предметов.

Заключение

В ходе выполнения бакалаврской работы был разработан программный модуль контроля целостности каталогов и файлов под UEFI, который успешно решает поставленные задачи. Модуль обеспечивает мониторинг объектов, поставленных на контроль, и оперативно реагирует на любые несанкционированные изменения, такие как модификация содержимого, переименование или удаление файлов и каталогов. В случае обнаружения нарушений программа блокирует загрузку операционной системы и информирует пользователя о характере нарушения, что способствует предотвращению потенциальных атак.

Разработанный модуль обладает модульной архитектурой, что обеспечивает гибкость и удобство масштабирования. Для реализации интерфейса использовалась библиотека LVGL, которая позволила создать интуитивно понятный и функциональный пользовательский интерфейс. Модуль поддерживает работу с различными файловыми системами и томами, а также обеспечивает сохранение результатов проверки в XML-файл.

Тестирование модуля подтвердило его корректную работу во всех предусмотренных сценариях, включая изменение названий файлов и каталогов, их удаление и модификацию содержимого. Программа успешно выявляет нарушения целостности и предоставляет подробную информацию о них.

В результате была достигнута цель работы — создан надежный инструмент для контроля целостности данных в UEFI-среде, который может быть интегрирован в системы доверенной загрузки. Разработанный модуль соответствует современным требованиям безопасности и может быть использован для защиты критически важных компонентов операционной системы.

Список использованных источников

1. Harrison, A. Harnessing the UEFI Shell: Moving the Platform Beyond DOS, Second Edition, 2019 – 416 с.

2. Mueller, V. Beyond BIOS: Developing with the Unified Extensible Firmware Interface, Third Edition., 2017 – 336 с.

3. Банк данных угроз безопасности информации [Электронный ресурс]: Режим доступа: http://www.bdu.fstec.ru/threat, свободный, (дата обращения 15.04.2025).

4. Немного про UEFI и Secure Boot [Электронный ресурс]: Режим доступа: https://habrahabr.ru/post/185492, свободный, (дата обращения 10.04.2025).

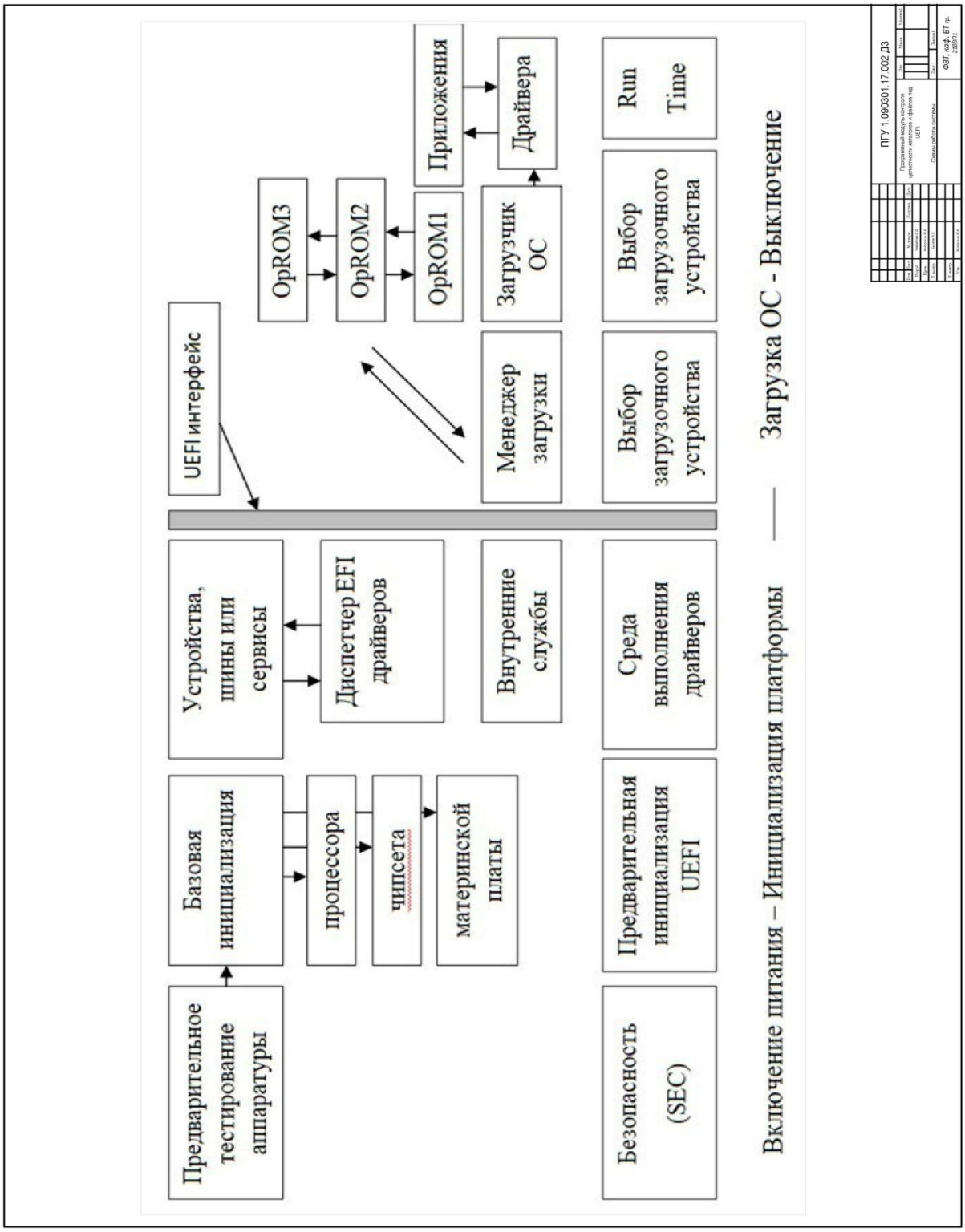
5. Разработка программной части программно-аппаратного комплекса на примере написания журнала событий (uefi bios) / Нефедова Е. Д., Митрохин М. А. // Журнал «Вестник Пензенского Государственного Университета», 2025 – 94 с.

Приложение А

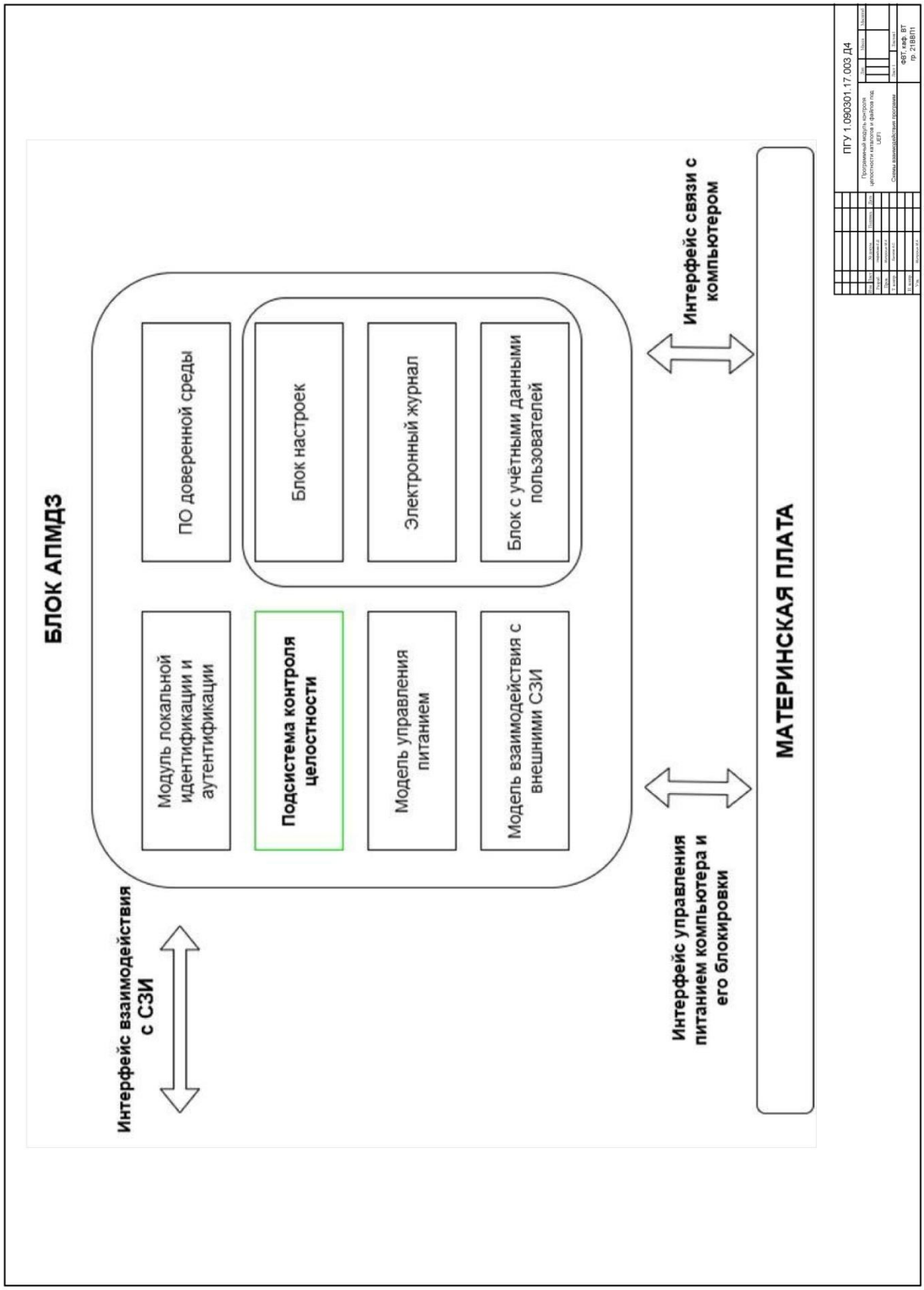
(Обязательное)

«Программный модуль контроля целостности каталогов и файлов под UEFI»

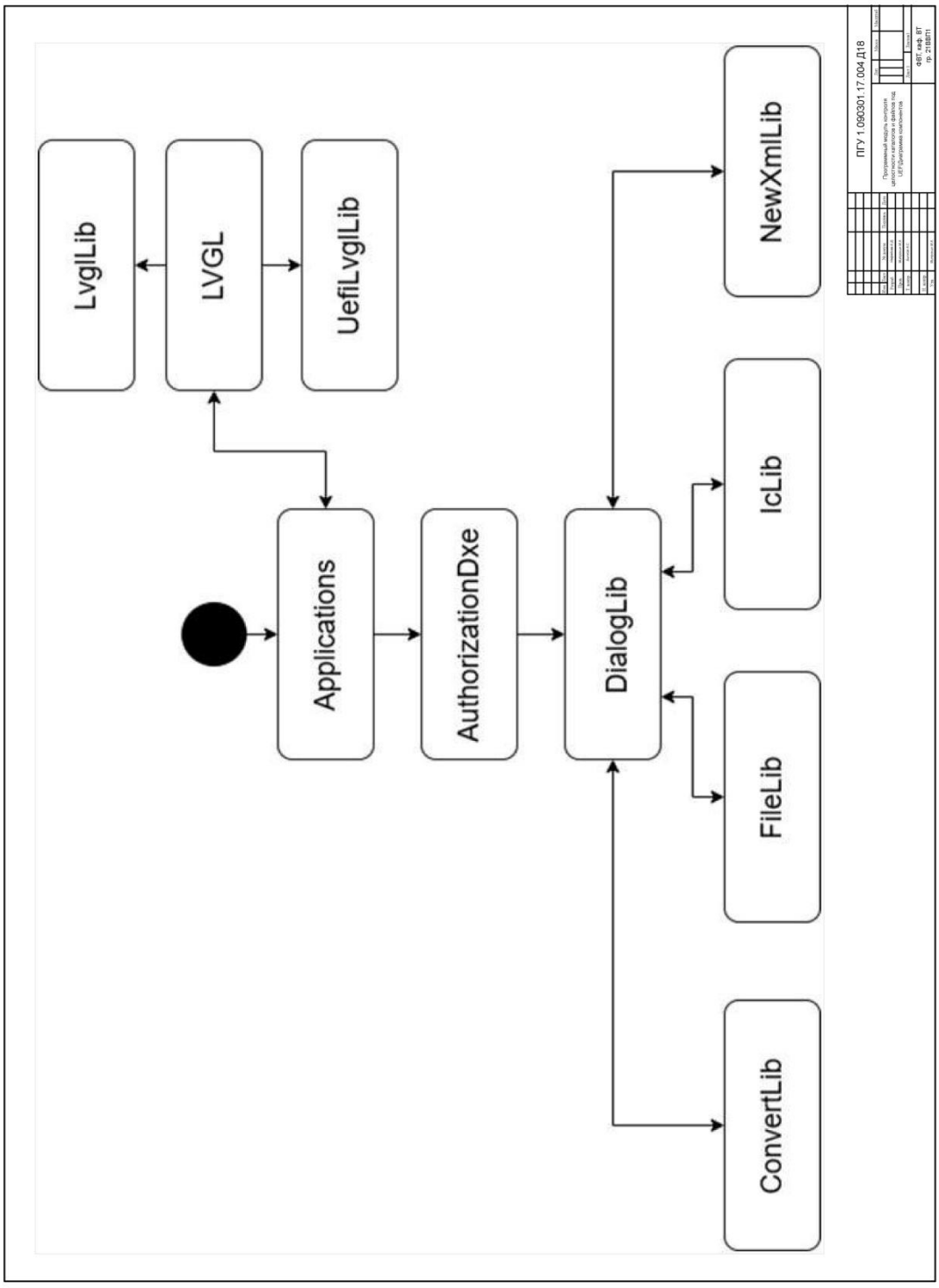
Приложение А.1 – Схема работы системы



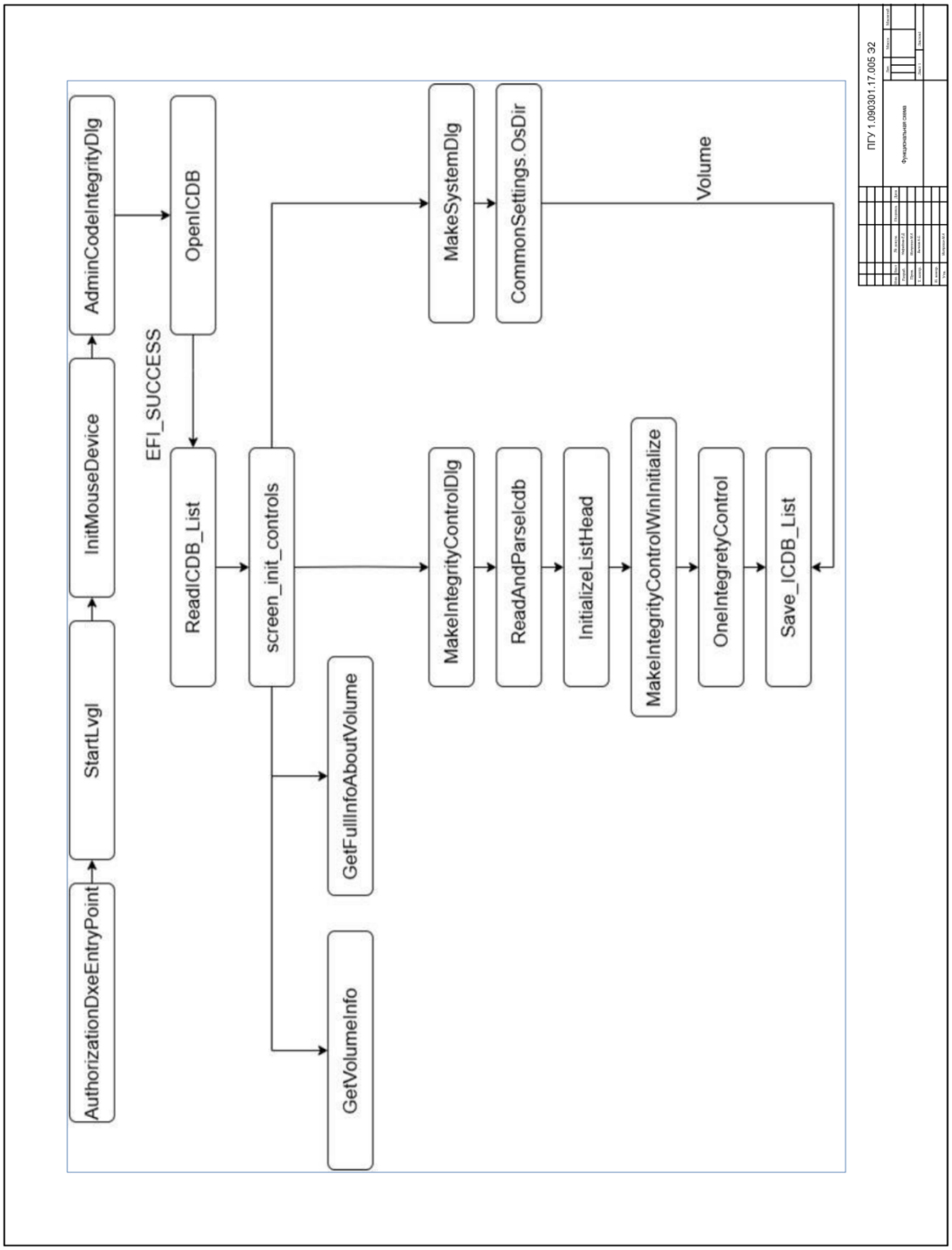
Приложение А.2 – Схема взаимодействия программ



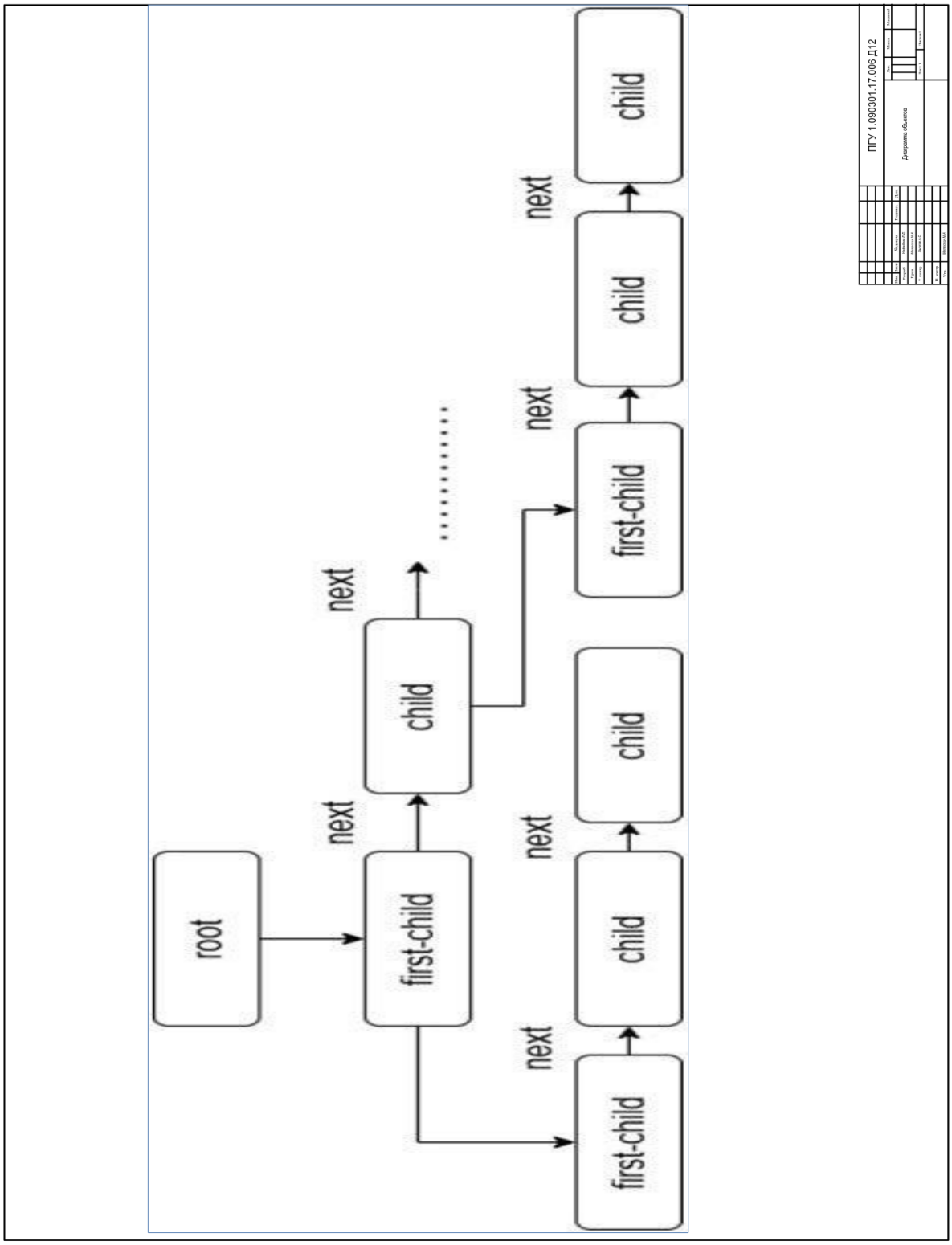
Приложение А.3 – Диаграмма компонентов



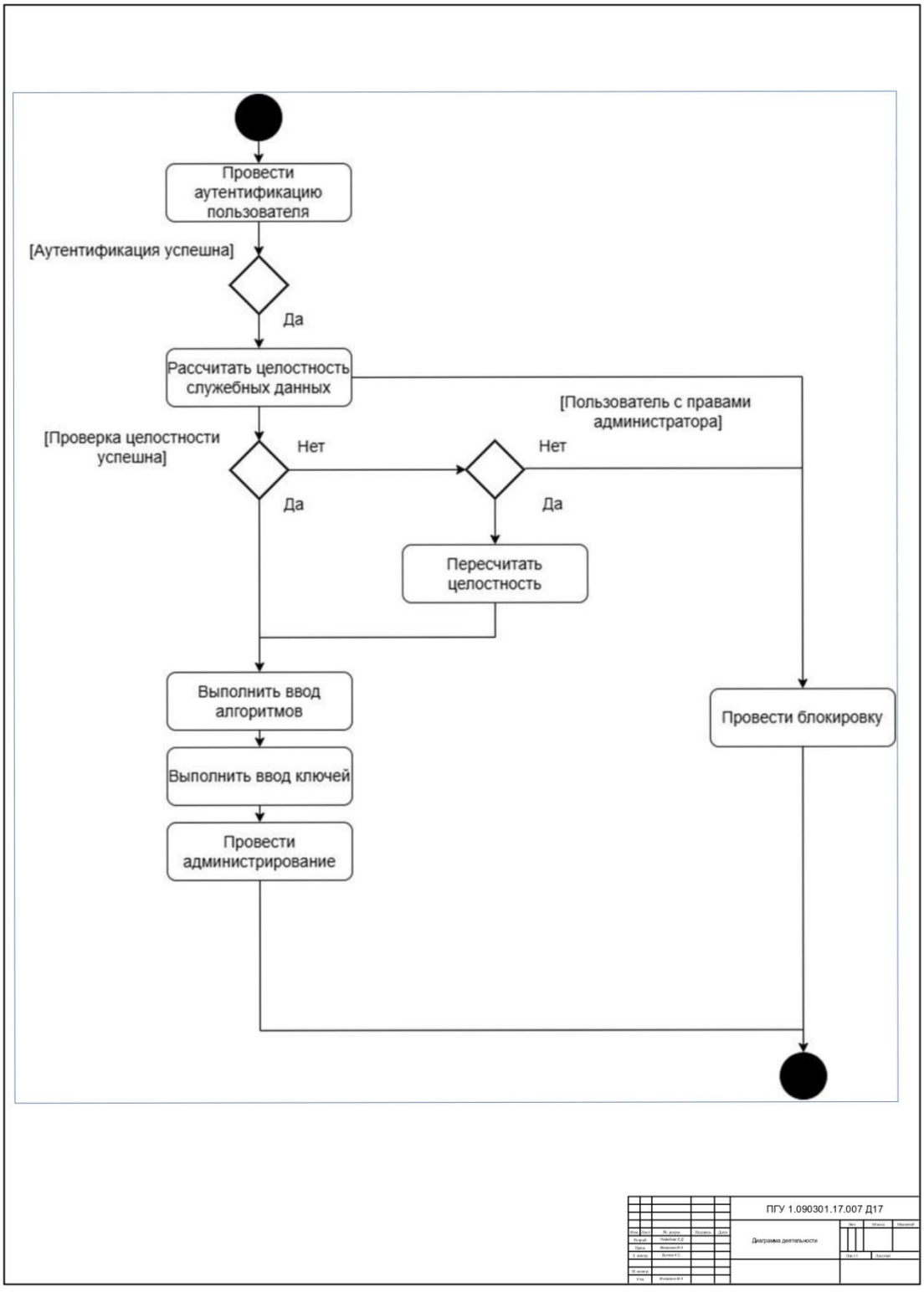
Приложение А.4 – Функциональная схема



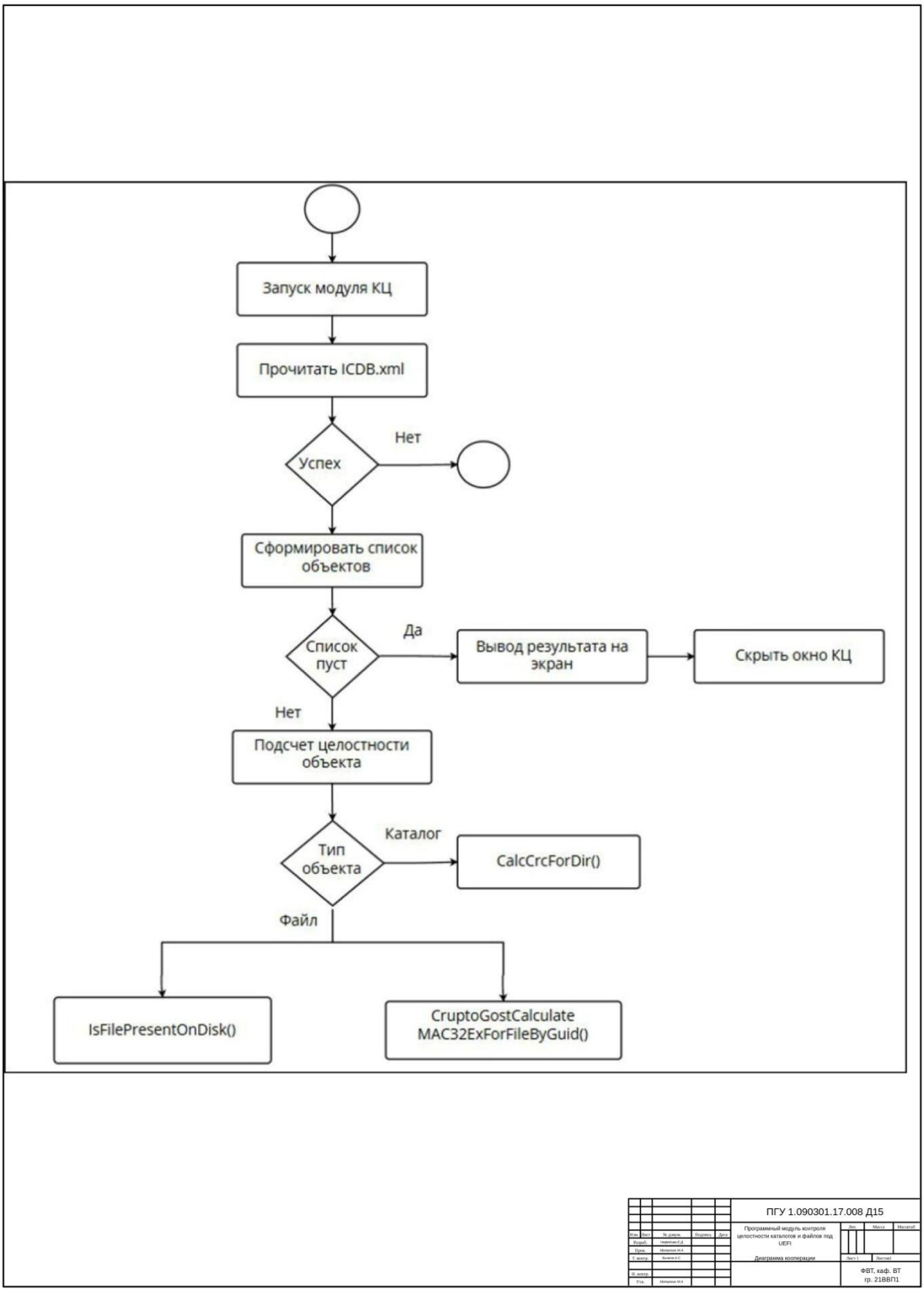
Приложение А.5 – Диаграмма объектов



Приложение А.6 – Диаграмма деятельности



Приложение А.7 – Диаграмма кооперации



ПГУ 1.090301.17.008 Д15

Программный модуль контроля целостности каталогов и файлов под UEFI

Диаграмма кооперации

ФВТ, каф. ВТ гр. 21ВВП1

Приложение В
(Обязательное)

«Программный модуль контроля целостности каталогов и файлов под UEFI»

Листинг программы

ФайлAuthorizationDxe.cpp

```
#include"StdAfx.h"


UINT16 g_LastScancode;

wchar_t szVersion[] = L"Версия: 0.0.1";
wchar_t szInfo[] = L"Информация";

#ifdef__cplusplus
extern"C" {
#endif
#ifdef__cplusplus
}
#endif

LARGE_INTEGER   gliAccurateTimeOffset;
 ///<СмещениевременихостаотносительноКП
BOOL            gbActualSystemTime;      ///< Признак получения
времени от КП


typedefstruct
{
 //зарезервировано
 UINT8 dbReservid[3];

 //структура с данными для модуля контроля загрузки
 //SBootControlParams BootControlParams;

} SBootControlParamsEx;

//EFI_GUID gEfiBootControlProtocolGuid =
EFI_BOOTCONTROL_PROTOCOL_GUID;

//----------------------------------------------------------------
-------------
/// <summary>
/// Функция выполнит перезапуск протоколов
/// </summary>
/// <returns>EFI_SUCCESS - еслиуспех</returns>
//----------------------------------------------------------------
-------------
EFI_STATUS ConnectAllImages()
{
 EFI_STATUS  efiStatus;
 UINTN       HandleCount;
 EFI_HANDLE* HandleBuffer;
 UINTN       Index;
```

```
 efiStatus = gBS->LocateHandleBuffer(
      ByProtocol,
      &gEfiBlockIoProtocolGuid,
      NULL,
      &HandleCount,
      &HandleBuffer
 );

 if (!EFI_ERROR(efiStatus))
 {
      for (Index = 0; Index < HandleCount; Index++)
      {
           efiStatus = gBS->ConnectController(HandleBuffer[Index],
(EFI_HANDLE*)NULL, (EFI_DEVICE_PATH_PROTOCOL*)NULL, FALSE);
      }

      if (HandleBuffer != NULL)
           FreePool(HandleBuffer);
 }

 return efiStatus;
}


STATIC
VOID
StopUefi()
{
#ifndef EMULATOR
 //while (TRUE)
 {
      //ждемминуту
      gBS->Stall(60 * 1000 * 1000);
 }
#endif
}

/**
  Производит поиск первого вхождения файла в корне всех разделов.

  @param[in]  FileName - Указатель на строку с именем файла.

  @retval Указатель на протокол файла, если файл найден. NULL
еслиненашелся.
**/
static
EFI_FILE_PROTOCOL*
FindFileByName(
 CONSTCHAR16* FileName,
 UINT64Mode
)
```

```
{
 EFI_FILE_PROTOCOL* desiredFileProtocol =
(EFI_FILE_PROTOCOL*)NULL;
 EFI_HANDLE* fsHandles = (EFI_HANDLE*)NULL;
 EFI_STATUS status;
 UINTN countFs;

 status = gBS->LocateHandleBuffer(ByProtocol,
&gEfiSimpleFileSystemProtocolGuid, NULL, &countFs, &fsHandles);

 if (EFI_ERROR(status))
 {
      return (EFI_FILE_PROTOCOL*)NULL;
 }

 for (UINTN index = 0; index < countFs; index++)
 {
      EFI_HANDLE fsController = fsHandles[index];
      EFI_SIMPLE_FILE_SYSTEM_PROTOCOL* simpleFileSystem =
(EFI_SIMPLE_FILE_SYSTEM_PROTOCOL*)NULL;

      status = gBS->OpenProtocol(
           fsController,
           &gEfiSimpleFileSystemProtocolGuid,
           (void**)&simpleFileSystem,
           gImageHandle,
           NULL,
           EFI_OPEN_PROTOCOL_BY_HANDLE_PROTOCOL);

      if (EFI_ERROR(status))
      {
           continue;
      }

      EFI_FILE_PROTOCOL* fsRootFile = (EFI_FILE_PROTOCOL*)NULL;
      EFI_FILE_PROTOCOL* fileProtocol = (EFI_FILE_PROTOCOL*)NULL;

      status = simpleFileSystem->OpenVolume(simpleFileSystem,
&fsRootFile);
      gBS->CloseProtocol(fsController,
&gEfiSimpleFileSystemProtocolGuid, gImageHandle, NULL);
      if (EFI_ERROR(status))
      {
           continue;
      }

      status = fsRootFile->Open(fsRootFile, &fileProtocol,
(CHAR16*)FileName, Mode, 0);

      fsRootFile->Close(fsRootFile);
```

```
        if (status == EFI_SUCCESS)
        {
             desiredFileProtocol = fileProtocol;
             break;
        }
    }

    FreePool(fsHandles);

    return desiredFileProtocol;
}


static
VOID
PressedScancodeHandler(VOID* Context, UINT16Scancode)
{
    LOG_INFO("Scancode: %c (%d)", Scancode, Scancode);
}

//-----------------------------------------------------------------
-----------
/// <summary>
/// Функция выполнит инициализацию стилей Lable14
/// </summary>
/// <returns>Стиль Lable14</returns>
//-----------------------------------------------------------------
-----------
lv_style_t*
DefaultLblStyle14(VOID)
{
    // Созданиестилядля Lable
    staticlv_style_t style_label = { 0 };

    if (style_label.sentinel != LV_STYLE_SENTINEL_VALUE)
    {
        lv_style_init(&style_label);
        lv_style_set_text_font(&style_label,
&lv_font_roboto_regular_14);
    }

    return&style_label;
}

//-----------------------------------------------------------------
-----------
/// <summary>
/// Функция выполнит инициализацию стилей Lable16
/// </summary>
/// <returns>Стиль Lable16</returns>
```

```
//-----------------------------------------------------------------
-----------
lv_style_t*
DefaultLblStyle16(VOID)
{
 // Созданиестилядля Lable
 staticlv_style_t style_label = { 0 };

 if (style_label.sentinel != LV_STYLE_SENTINEL_VALUE)
 {
      lv_style_init(&style_label);
      lv_style_set_text_font(&style_label,
&lv_font_roboto_regular_16);
 }

 return&style_label;
}

//-----------------------------------------------------------------
-----------
/// <summary>
/// Функция выполнит инициализацию стилей Lable30
/// </summary>
/// <returns>Стиль Lable16</returns>
//-----------------------------------------------------------------
-----------
lv_style_t*
DefaultLblStyle30(VOID)
{
 // Созданиестилядля Lable
 staticlv_style_t style_label = { 0 };

 if (style_label.sentinel != LV_STYLE_SENTINEL_VALUE)
 {
      lv_style_init(&style_label);
      lv_style_set_text_font(&style_label,
&lv_font_roboto_regular_30);
 }

 return&style_label;
}

//-----------------------------------------------------------------
-----------
/// <summary>
/// Функция подготовит стиль кнопок по умолчанию
/// </summary>
/// <returns>стилькнопокпоумолчанию</returns>
//-----------------------------------------------------------------
---------
lv_style_t* DefaultBtnStyle14(VOID)
```

```
{
 // Созданиестилядлякнопок
 staticlv_style_t style_btn = { 0 };

 if (style_btn.sentinel != LV_STYLE_SENTINEL_VALUE)
 {
      lv_style_init(&style_btn);
      lv_style_set_bg_color(&style_btn,
lv_palette_main(LV_PALETTE_BLUE));
      lv_style_set_border_width(&style_btn, 3);
      lv_style_set_text_font(&style_btn,
&lv_font_roboto_regular_14);
 }

 return&style_btn;
}

//-----------------------------------------------------------------
-----------
/// <summary>
/// Вторая точка входа в приложение авторизации. Необходима если
запускаем приложение с новым стеком. Вызывается из основной точки
входа.
/// </summary>
/// <param Name="Context1">Указатель на первый контекст.
Неиспользуемый параметр.</param>
/// <param Name="Context2">Указатель на второй контекст.
Неиспользуемыйпараметр.</param>
/// <returns>EFI_SUCCESS - еслиуспех</returns>
//-----------------------------------------------------------------
---------
static
VOID
AuthorizationDxeEntryPoint(VOID* Context1, VOID* Context2)
{
 EFI_GRAPHICS_OUTPUT_PROTOCOL* graphicsOutput;
 EFI_STATUS status;

 //версияэкстендора
 //LOG_INFO("Authorization module. CRYPTOSOFT. 2023. %S",
GetVersionInfoStr());

 //получение информации связанных с томами
 //status = GetFullInfoAboutVolume(0, (PFULL_VOLUME_INFO)NULL);
 //gBS->Stall(15 * 1000 * 1000);

 status = gBS->LocateProtocol(&gEfiGraphicsOutputProtocolGuid,
NULL, (VOID**)&graphicsOutput);

 if (EFI_ERROR(status))
 {
```

```
      LOG_ERROR("LocateProtocol(gEfiGraphicsOutputProtocolGuid),
Status: 0x%08X", status);
      StopUefi();
 }

 status = StartLvgl(graphicsOutput, gST->ConIn,
PressedScancodeHandler, NULL);

 if (EFI_ERROR(status))
 {
      LOG_ERROR("StartLvgl(), Status: 0x%08X", status);
      StopUefi();
 }

 //инициализируеммышь
 status = InitMouseDevice(graphicsOutput);

 //запуск админки КЦ для удобной отладки
 AdminCodeIntegrityDlg();
 }

//класс загрузки необходимых драйверов
classCQpDriversList
{
private:
 typedefstruct_QP_DRIVERS
 {
      CHAR16* szDriversName;
      EFI_STATUS StartStatus;
 } QP_DRIVERS;

#ifdef EMULATOR
 //массивдрайверов
 QP_DRIVERS QpDrivers[2] = {
      (CHAR16*)L"PromDxe.efi", EFI_LOAD_ERROR,
      (CHAR16*)L"RomFs.efi", EFI_LOAD_ERROR };
#else
 QP_DRIVERS QpDrivers[1] = {
 (CHAR16*)L"RomFs.efi", EFI_LOAD_ERROR};
#endif

public:
 EFI_STATUS StartQpDrivers();
};

//метод класса выполнит запуск всех необходимых драйверов
EFI_STATUSCQpDriversList::StartQpDrivers()
{
 EFI_STATUS efiStatus = EFI_SUCCESS;
```

```
 for (UINTN i = 0; i <sizeof(QpDrivers) / sizeof(QpDrivers[0]);
i++)
 {
      //если драйвер еще не запущен то запускаем
      if (QpDrivers[i].StartStatus != EFI_SUCCESS)
      {
           QpDrivers[i].StartStatus = StartModule(gImageHandle,
QpDrivers[i].szDriversName);
           LOG_ERROR("StartModule(%s). Status: 0x%X",
QpDrivers[i].szDriversName, QpDrivers[i].StartStatus);
      }
 }

 LOG_INFO("efiStatus: 0x%08X", efiStatus);

 return efiStatus;
}

/**
  Точка входа в приложение AuthorizationDxe.

  @param[in]  ImageHandle - Указатель на описатель образа модуля.
  @param[in]  SystemTable - Указатель на системную таблицу с
функциями.

@retval Статусвыполнения.
**/
EFI_STATUS
UefiMain(INEFI_HANDLEImageHandle,
 INEFI_SYSTEM_TABLE* SystemTable)
{
 LOG_INFO("");

 gBS->SetWatchdogTimer(0, 0, 0, (CHAR16*)NULL);

 //пример использование класса
 CQpDriversList QpDriversClass;
 QpDriversClass.StartQpDrivers();

 AuthorizationDxeEntryPoint(NULL, NULL);

 // Сюданеходим.
 returnEFI_SUCCESS;}
AdminCodeIntegrity.c
#include <Uefi.h>
#include <Library/UefiLib.h>
#include <Library/UefiBootServicesTableLib.h>
#include <Library/ConvertLib.h>
#include <Library/PrintLib.h>
#include <Library/MemoryAllocationLib.h>
```

```
#include <Library/BaseMemoryLib.h>
#include <Library/UefiLvglLib.h>
#include <Library/LvglLib.h>
#include <Guid/FileSystemInfo.h>

#include <Library/DialogLib.h>

#define LABEL_UP_TIP " END - перемещениена \"Изменитьпуть\"."
#define LABEL_DOWN_TAB_TIP " TAB - перемещение вниз по списку."
#define LABEL_ENTER_TIP " ENTER - показать каталоги/файлы
выбранного тома."
#define WRAPPER_WIDTH LV_PCT(100)
#define WRAPPER_HEIGHT LV_PCT(100)
#define WRAPPER_GAP (5)

#define WRAPPER_WIDTH LV_PCT(100)
#define WRAPPER_HEIGHT LV_PCT(100)
#define WRAPPER_GAP (5)

#define FORM_WIDTH (1000) //ширинацентральнойформы
#define FORM_HEIGHT (820) //высота центральной формы

#define HEADER_WIDTH (1000)//длина верхнего контейнера (контроль
файлов)
#define HEADER_HEIGHT (50)//ширина верхнего контейнера (контроль
файлов)

#define POPUP_WIDTH (1000)//ширина контейнера с информацией по
навигации
#define POPUP_HEIGHT (80)//высота контейнера с информацией по
навигации

#define FORM_VERTICAL_PADDING (0)//сдвиг контейнеров вниз от
рамки (центр-левый)
#define FORM_HORIZONTAL_PADDING (0)//сдвиг контейнеров вправо от
рамки (центр-левый)
#define FORM_VERTICAL_PADDING_L (0)//сдвиг контейнеров вниз от
рамки (центр-левый)
#define FORM_HORIZONTAL_PADDING_L (0)//сдвиг контейнеров вправо
от рамки (центр-левый)
#define FORM_VERTICAL_PADDING_R (0)//сдвиг контейнеров вниз от
рамки (центр-правый)
#define FORM_HORIZONTAL_PADDING_R (0)//сдвиг контейнеров вправо
от рамки (центр-правый)
#define FORM_RADIUS (10)//скруглениерамки
#define FORM_BORDER_WIDTH (0)//ширинарамки
#define FORM_GAP (5)//отступ между контейнерами

#define FORM_GAP_CTR_DOWN (10)//отступ между контейнерами
#define FORM_GAP_CTR_UP (10)//отступ между контейнерами на
центральной форме
```

```
#define FORM_GAP_STR (1)//отступ между строками в контейнере

#define NOTIFICATION_WIDTH FORM_WIDTH
#define NOTIFICATION_HEIGHT (60)
#define NOTIFICATION_PADDING (0)
#define NOTIFICATION_RADIUS (7)
#define NOTIFICATION_BORDER_WIDTH (3)

#define FORM_CONT_PADDING (0)
#define FORM_CONT_RADIUS (10)
#define FORM_CONT_WIDTH LV_PCT(100)
#define FORM_CONT_BORDER_WIDTH (3)//рамка у верхнего и нижнего
контейнеров
#define FORM_CONT_BORDER_COLOR
(lv_palette_main(LV_PALETTE_GREEN))

//#define TITLE_HEIGHT LV_PCT(130)
//#define BODY_GAP (0)

#define TEXTAREA_INFO_HEIGHT (40)
#define TIPS_CONT_HEIGHT (90)
#define TITLE_LABELS_GAP (0)
#define TIPS_LABELS_GAP (0)
#define INPUTS_GAP (10)
#define INPUTS_PADDING (5)
#define NOTIFY_WIDGET_GAP (10)

#define TEXTAREA_INPUT_LOGIN_MAX_LENGTH 16
#define TEXTAREA_INPUT_PASSWORD_MAX_LENGTH 16
#define TEXTAREA_INPUT_HEIGHT (30)
#define TEXTAREA_INPUT_WIDTH LV_PCT(100)
#define TEXTAREA_INPUT_RADIUS (4)
#define TEXTAREA_INPUT_PADDING_TOP (6)
#define TEXTAREA_INPUT_PADDING_LEFT (80)

#define DISK_PANEL_WIDTH  (482)
#define DISK_PANEL_HEIGHT (200)


// Сигнал для сворачивания графической подсистемы. Сигнальное
состояние != 0.
STATIC UINT8 g_QuitSignal = 0;

//статус выхода из диалога (по умолчанию признак нажатия кнопки
"Отмена")
UINT8 g_ButtonID = 0;


//tree_value* g_pFirstFS;

LV_IMG_DECLARE(lv_img_warning)
```

```
LV_IMG_DECLARE(lv_img_cryptosoft)
LV_IMG_DECLARE(lv_pac_m_logo_bg)
LV_IMG_DECLARE(lv_pac_m_logo_main)

/**
  @typedef screens_enum
  @brief   Экраны.
**/
typedef enum _screens_type_t
{
 base_screen,
 screen_initialization_mode,
 screen_authorization
}screens_type_t;

/**
  @struct _form_styles_t
  @brief  Таблица со всеми стилями для контролов экрана.
**/
typedef struct _form_styles_t
{
 lv_style_t label_30;
 lv_style_t label_16;
 lv_style_t label_14;
}form_styles_t;

lv_obj_t* cont_form_down;

/**
  @struct _screen_attr_t
  @brief  Описание элементов к которым необходимо иметь доступ.
**/
typedef struct _screen_attr_t
{
 // Текущийактивныйэкран.
 screens_type_t state;

}screen_attr_t;

// Элементы к которым необходимо иметь доступ.
static screen_attr_t g_attr_screens;
//обработка событий внутри списка с каталогами и файлами
static void event_handler_file(lv_event_t* e);
//Обработка перехода из одного контейнера в другой
static void event_handler_change(lv_event_t* e);
static void event_write_List_IC(lv_event_t* e);
//обработка событий внутри списка с томами
void event_handler_directory(lv_event_t* e);
void enter_volume_mode_handler();
void event_handler_control(lv_event_t* e);
```

```
//Главный диалог КЦ
lv_obj_t* g_MainIcDlg;

char* text_utf8_up_tip;
char* text_utf8_down_tip;
char* text_utf8_enter_tip;
char* text_utf8_disk_file_cntrl;
extern EFI_GRAPHICS_OUTPUT_PROTOCOL* graphicsOutput;

static lv_obj_t* list_tom;
static lv_obj_t* list_control;
static lv_obj_t* list_file_and_dir;
lv_obj_t* label_up_tip;
lv_obj_t* label_down_tip;
lv_obj_t* label_enter_tip;

static lv_style_t form_cont_left;
static lv_obj_t* cont_form_l;
static lv_obj_t* cont_right_panel;
static lv_style_t cb_cont_left;
static lv_style_t bg;
static lv_style_t form_cont_right;
bool res;
UINTN count_control = 0;
lv_obj_t* cont_form_popup;

lv_obj_t* focused_button;

//максимальное число файловых систем поддерживаемых программой
#define MAX_FS_COUNT (20)

static efs_info efs_infos[MAX_FS_COUNT] = { 0 };
static unsigned long efs_count = 0;
static lv_obj_t* checkboxes;
lv_obj_t* btn_update;
lv_obj_t* btn_textarea;
static lv_obj_t* bt;
static lv_obj_t* bt_control;
static lv_obj_t* label_Label;
static lv_obj_t* label_file_Attribute;
static lv_obj_t* label_file_CreateTime;
static lv_obj_t* label_file_FileName;
static lv_obj_t* label_file_FileSize;
static lv_obj_t* label_file_LastAccessTime;
static lv_obj_t* label_file_ModificationTime;
static lv_obj_t* label_file_PhysicalSize;
static lv_obj_t* label_fileInfo;
static lv_obj_t* label_VolumeLabel;
static lv_obj_t* label_VolumeGuid;
static lv_obj_t* label_VolumeSize;
static lv_obj_t* label_FreeSpace;
```

```
static lv_obj_t* label_FSType;
static lv_obj_t* label_SerialNumber;
static lv_obj_t* label_BlockSize;
static lv_obj_t* label_ReadOnly;
static lv_obj_t* label_Files;
static lv_obj_t* label_Dirs;
static lv_obj_t* cont_form_down_3;
efs_info* curr_volume = NULL;
static direct_info* cur_dir_info;

//Функция переводит  в ниболее подходящую единицу измерения
void method(lv_obj_t* label, UINTN Size, char* str)
{
 UINTN Size_first = Size / 1024; //КБ
 UINTN Size_second = Size % 1024;
 char buffer[500];
 if (Size_first >= 1024)
 {
      Size_first = Size_first / 1024;//МБ file_FileSize/1024/1024
      Size_second = (Size / 1024) % 1024;
      if (Size_first >= 1024)
      {
           Size_first = Size_first / 1024;//ГБ
file_FileSize/1024/1024/1024
           Size_second = (Size / 1024 / 1024) % 1024;
           if (Size_first >= 1024)
           {
                Size_first = Size_first / 1024;//ТБ
file_FileSize/1024/1024/1024/1024
                Size_second = (Size / 1024 / 1024 / 1024) % 1024;
                lv_snprintf(buffer, sizeof(buffer), " %s: %d,%d Тб
\n", str, Size_first, Size_second);
                lv_label_set_text(label,
ConvertStrToUtf8(buffer));
           }
           else
           {
                lv_snprintf(buffer, sizeof(buffer), " %s: %d,%d Гб
\n", str, Size_first, Size_second);
                lv_label_set_text(label,
ConvertStrToUtf8(buffer));
           }
      }
      else
      {
           lv_snprintf(buffer, sizeof(buffer), " %s: %d,%d Мб \n",
str, Size_first, Size_second);
           lv_label_set_text(label, ConvertStrToUtf8(buffer));
      }
 }
 else
```

```
 {
      lv_snprintf(buffer, sizeof(buffer), " %s: %d Байт \n", str,
Size);
      lv_label_set_text(label, ConvertStrToUtf8(buffer));
 }
}

void event_handler_for_btn_IC(lv_event_t* e)
{
 EFI_STATUS efi_status;
 EFI_FILE_PROTOCOL* icdbFile = (EFI_FILE_PROTOCOL*)NULL;
 lv_event_code_t code = lv_event_get_code(e);

 if (code == LV_EVENT_KEY)
 {
      char c = *((char*)lv_event_get_param(e));
      if (LV_KEY_END == c)
      {
           //lv_group_focus_obj(btn_update);
           lv_group_focus_obj(btn_textarea);
      }
 }
}

lv_obj_t* Create_list_file_new(tree_value* child_value)
{
 lv_obj_t* btn = lv_list_add_btn(list_file_and_dir,
LV_SYMBOL_FILE, Uint16ToUint8(child_value->name));
 lv_obj_add_event_cb(btn, event_handler_file, LV_EVENT_ALL,
child_value);
 lv_obj_add_event_cb(btn, event_handler_change, LV_EVENT_ALL,
child_value);
 lv_obj_add_event_cb(btn, event_handler_for_btn_IC, LV_EVENT_ALL,
child_value);
 lv_obj_add_event_cb(btn, event_write_List_IC, LV_EVENT_KEY,
child_value);
 return btn;
}

lv_obj_t* Create_list_dir_new(tree_value* child_value)
{
 lv_obj_t* btn = lv_list_add_btn(list_file_and_dir,
LV_SYMBOL_DIRECTORY, Uint16ToUint8(child_value->name));
 lv_obj_add_event_cb(btn, event_handler_directory, LV_EVENT_ALL,
child_value);
 lv_obj_add_event_cb(btn, event_handler_change, LV_EVENT_ALL,
child_value);
 lv_obj_add_event_cb(btn, event_handler_for_btn_IC, LV_EVENT_ALL,
child_value);
 lv_obj_add_event_cb(btn, event_write_List_IC, LV_EVENT_KEY,
child_value);
```

```
 return btn;
}

//Создание checkbox
lv_obj_t* Create_checkbox(lv_obj_t* btn)
{
 lv_obj_t* checkbox = lv_checkbox_create(btn);
 lv_checkbox_set_text(checkbox, "");
 lv_group_remove_obj(checkbox);
 lv_obj_add_style(checkbox, &cb_cont_left, LV_PART_MAIN |
LV_PART_INDICATOR | LV_STATE_DEFAULT);
 lv_obj_add_style(checkbox, &cb_cont_left, LV_PART_MAIN |
LV_PART_INDICATOR | LV_STATE_CHECKED);
 lv_obj_add_state(checkbox, LV_STATE_DEFAULT);
 return checkbox;
}

//Создание листа с файлами на КЦ
lv_obj_t* Create_list_control(lv_obj_t* form_list) {
 list_control = lv_list_create(form_list);
 lv_obj_set_size(list_control, 486, 164);
 lv_obj_center(list_control);
 return list_control;
}

//Создание labels для описания навигации по модулю
lv_obj_t* Create_labels(lv_obj_t* label, char* text_utf8) {
 label = lv_label_create(cont_form_popup);
 lv_obj_add_style(label, DefaultLblStyle14(), LV_STATE_DEFAULT);
 lv_label_set_text(label, text_utf8);
 return label;
}

//Создание labels с атрибутами тома
void Create_VolumeAttribute(lv_obj_t** label_volume_attribute)
{
 *label_volume_attribute = lv_label_create(label_Label);
 lv_obj_add_style(*label_volume_attribute, DefaultLblStyle16(),
LV_STATE_DEFAULT);
 return;
}

static BOOLEAN
FindFileInXml(CHAR16* data_file, tree_value* child) {

 EFI_STATUS efi_status;
 EFI_FILE_PROTOCOL* icdbFile = (EFI_FILE_PROTOCOL*)NULL;
 IC_FILE_INFO* Ic_Info =
(IC_FILE_INFO*)AllocateZeroPool(sizeof(IC_FILE_INFO));
 CHAR8* pBuffer = NULL;
 UINTN BufferSize = 0;
```

```
 efi_status = OpenICDB(EFI_FILE_MODE_WRITE | EFI_FILE_MODE_READ,
&icdbFile);
 if (EFI_ERROR(efi_status))
 {
      return FALSE;
 }
 CHAR16* szControlFileName = NULL;
 UINTN StrLenFileName;
 CHAR16* FullNameVolume;
 Ic_Info->pFileName = (CHAR16*)data_file;

 FULL_VOLUME_INFO FullInfo;
 //efi_status = GetFullInfoAboutVolume(curr_volume->fs_index,
&FullInfo);
 efi_status = GetFullInfoAboutVolume(child->efs->fs_index,
&FullInfo);
 EFI_GUID GUID = FullInfo.PartitionGuid;
 //CHAR16* str_guid = GuidToStrW(&GUID);
 CHAR16 Volume_str[0x60];
 efi_status = UnicodeSPrint(Volume_str, sizeof(Volume_str),
L"\\\\.\\Volume{%08x-%04x-%04x-%02x%02x-
%02x%02x%02x%02x%02x%02x}", (UINTN)GUID.Data1, (UINTN)GUID.Data2,
(UINTN)GUID.Data3, (UINTN)GUID.Data4[0], (UINTN)GUID.Data4[1],
(UINTN)GUID.Data4[2], (UINTN)GUID.Data4[3], (UINTN)GUID.Data4[4],
(UINTN)GUID.Data4[5], (UINTN)GUID.Data4[6],
(UINTN)GUID.Data4[7]);

 if (child->parent == child->efs->root)
 {
      StrLenFileName = StrLen(Ic_Info->pFileName) + 1 + 1;//слеш +
терминальныйноль
      szControlFileName = (CHAR16*)AllocateZeroPool(StrLenFileName
* sizeof(CHAR16));
      StrCpyS(szControlFileName, StrLenFileName, L"\\");
      StrCatS(szControlFileName, StrLenFileName, Ic_Info-
>pFileName);

 }
 else
 {
      StrLenFileName = StrLen(child->parent->name) +
StrLen(Ic_Info->pFileName) + 1;//терминальныйноль
      szControlFileName = (CHAR16*)AllocateZeroPool(StrLenFileName
* sizeof(CHAR16));
      StrCpyS(szControlFileName, StrLenFileName, child->parent-
>name);
      StrCatS(szControlFileName, StrLenFileName, Ic_Info-
>pFileName);
 }
```

```
 UINTN tempFull = StrLen(szControlFileName) + StrLen(Volume_str)
+ 1;//терминальныйноль
 FullNameVolume = (CHAR16*)AllocateZeroPool(tempFull *
sizeof(CHAR16));
 ZeroMem(FullNameVolume, tempFull * sizeof(CHAR16));
 StrCatS(FullNameVolume, tempFull, Volume_str);
 StrCatS(FullNameVolume, tempFull, szControlFileName);

 BOOLEAN result = CompareNode(icdbFile, Ic_Info, FullNameVolume,
&pBuffer, &BufferSize);
 efi_status = icdbFile->Close(icdbFile);

 return result;
}


//Функция по перечислению директорий
void DirListCycle(tree_value* child_value)
{

 tree_value** p_child = &child_value->first_child;

 EFI_FILE_SYSTEM_INFO* info = (EFI_FILE_SYSTEM_INFO*)NULL;
 CHAR16* fileList = (CHAR16*)NULL;
 UINTN fileListSize = 0;
 CHAR16* dirList = (CHAR16*)NULL;
 UINTN dirListSize = 0;
 UINTN dwSize = 0;

 //если не заполнена структура
 if (*p_child == NULL)
 {

      EFI_STATUS Status = GetVolumeInfo(gImageHandle,
gImageHandle, child_value->efs->fs_index, info, &dwSize,
&fileList, &fileListSize, &dirList, &dirListSize, false,
(CHAR16*)NULL);
      _DEBUG_PRINT(DEBUG_INFO, "\nERROR:GetVolumeInfo(2)");
      info = (EFI_FILE_SYSTEM_INFO*)AllocateZeroPool(dwSize);

      _DEBUG_PRINT(DEBUG_WARN, "event_handler_directory : Trying
to open directory [%s].\n", child_value->Full_name);

      Status = GetVolumeInfo(gImageHandle, gImageHandle,
child_value->efs->fs_index, info, &dwSize, &fileList,
&fileListSize, &dirList, &dirListSize, true, child_value-
>Full_name);
      _DEBUG_PRINT(DEBUG_INFO, "\nERROR:GetVolumeInfo(3)");
      if (EFI_ERROR(Status))
      {
```

```
            lv_label_set_text(label_VolumeSize,
ConvertStrToUtf8("Error while getting volume info with
directories and files"));
            return;
      }

      CHAR16* dirListTmp = dirList;
      CHAR16* fileListTmp = fileList;

      //если нет каталогов
      if (dirListTmp != NULL)
      {
            //цикл по всему буферу пока указатель dirListTmp не
вышел за границу буфера dirList
            while (dirListTmp < dirList + dirListSize)
            {
                  tree_value* child =
(tree_value*)AllocateZeroPool(sizeof(tree_value));
                  CHAR16* dir_name =
(CHAR16*)AllocateZeroPool(StrLen((CHAR16*)L"\\") * sizeof(CHAR16)
+ StrLen(dirListTmp) * sizeof(CHAR16) + sizeof(CHAR16));

                  memcpy(dir_name, L"\\", StrLen((CHAR16*)L"\\") *
sizeof(CHAR16));
                  memcpy(dir_name + StrLen((CHAR16*)L"\\"),
dirListTmp, StrLen(dirListTmp) * sizeof(CHAR16) +
sizeof(CHAR16));

                  child->type = 0;
                  child->efs = child_value->efs;
                  child->name = dir_name;
                  child->parent = child_value;

                  UINTN StrLenFileName= StrLen(child_value-
>Full_name) + StrLen(dirListTmp) + 1;//терминальныйноль
                  CHAR16* dir_name_full =
(CHAR16*)AllocateZeroPool(StrLenFileName * sizeof(CHAR16));
                  StrCpyS(dir_name_full, StrLenFileName,
child_value->Full_name);
                  StrCatS(dir_name_full, StrLenFileName,
dirListTmp);

                  child->Full_name = dir_name_full;

                  *p_child = child;
                  p_child = &(*p_child)->next;
                  //ставим указатель на следующее имя файла,
прибавляем к текущему указателю длинну строки + ноль-символ
                  dirListTmp += StrLen(dirListTmp) + 1;
            }
```

```
        }

        //если нет каталогов
        if (fileListTmp != NULL)
        {
            //цикл по всему буферу пока указатель dirListTmp не
вышел за границу буфера dirList
            while (fileListTmp < fileList + fileListSize)
            {
                tree_value* child =
(tree_value*)AllocateZeroPool(sizeof(tree_value));

                child->type = 1;
                child->efs = child_value->efs;
                child->name = fileListTmp;
                child->parent = child_value;

                *p_child = child;
                p_child = &(*p_child)->next;
                //ставим указатель на следующее имя файла,
прибавляем к текущему указателю длинну строки + ноль-символ
                fileListTmp += StrLen(fileListTmp) + 1;
            }
        }
    }

    tree_value* p_child_next = child_value->first_child;

    while (p_child_next)
    {
        if (p_child_next->type == 0)
        {

            bt = Create_list_dir_new(p_child_next);
            checkboxes = Create_checkbox(bt);

            //отладочная печать имени директории, который добавили
в интерфейс пользователя
            LOG_INFO_DLG_LIB("Creating directory button [%s], size
= %d.", p_child_next->name, StrLen(p_child_next->name));
        }
        else
        {
            bt = Create_list_file_new(p_child_next);
            checkboxes = Create_checkbox(bt);

            if (FindFileInXml(p_child_next->name, p_child_next) ==
TRUE)
            {
                lv_obj_add_state(checkboxes, LV_STATE_CHECKED);
```

```
            }
            else
            {
                 lv_obj_add_state(checkboxes, LV_STATE_DEFAULT);
            }

            //отладочная печать имени файла который добавили в
интерфейс пользователя
            LOG_INFO_DLG_LIB("Creating file button [%s], size =
%d.", p_child_next->name, StrLen(p_child_next->name));
      }
      p_child_next = p_child_next->next;
 }
}


//Функцияпоперечислениюфайлов
void FileListCycle(tree_value* child) {

 while (child)
 {
      bt = Create_list_file_new(child);
      checkboxes = Create_checkbox(bt);

      if (FindFileInXml(child->name, child) == TRUE)
      {
           lv_obj_add_state(checkboxes, LV_STATE_CHECKED);
      }
      else
      {
           lv_obj_add_state(checkboxes, LV_STATE_DEFAULT);
      }

      //отладочная печать имени файла который добавили в интерфейс
пользователя
      LOG_INFO_DLG_LIB("Creating file button [%s], size = %d.",
child->name, StrLen(child->name));

      //ставим указатель на следующее имя файла, прибавляем к
текущему указателю длинну строки + ноль-символ
      child = child->next;
 }
}

//Обработка нажатия на кнопку КЦ
static void event_handler_change_path(lv_event_t* e)
{
 lv_event_code_t code = lv_event_get_code(e);
 if (code == LV_EVENT_CLICKED)
 {
      //MakeChangePathDlg(graphicsOutput);
```

```
      MakeSystemDlg(/*graphicsOutput*/);
 }
}

//Обработканажатиянакнопку КЦ
static void event_handler_clicked(lv_event_t* e)
{
 EFI_STATUS efi_status;
 EFI_FILE_PROTOCOL* icdbFile = (EFI_FILE_PROTOCOL*)NULL;
 lv_event_code_t code = lv_event_get_code(e);
 if (code == LV_EVENT_CLICKED)
 {
      //LV_LOG_USER("Start event: ");
      /*CHAR8* pBuffer = NULL;
      UINTN BufferSize = 0;
      IC_FILE_INFO* IC_Info =
(IC_FILE_INFO*)AllocateZeroPool(sizeof(IC_FILE_INFO));*/
      //MakeJournalDlgWinInitializeNew(graphicsOutput);
      efi_status = OpenICDB(EFI_FILE_MODE_WRITE |
EFI_FILE_MODE_READ, &icdbFile);
      efi_status = Save_ICDB_List(icdbFile);
      efi_status = icdbFile->Close(icdbFile);

      g_QuitSignal = TRUE;
      g_ButtonID = 1;
 }
}

//Обработка перехода из одного контейнера в другой
static void event_handler_change(lv_event_t* e)
{
 lv_event_code_t code = lv_event_get_code(e);
 if (code == LV_EVENT_KEY)
 {
      char c = *((char*)lv_event_get_param(e));
      if (LV_KEY_RIGHT == c)
      {
           lv_obj_t* obj = lv_obj_get_child(list_control, 0);
           lv_group_focus_obj(obj);
      }
      else if (LV_KEY_LEFT == c)
      {
           if (focused_button)
                lv_group_focus_obj(focused_button);
      }
 }
}

//обработчик записи файла/директории в лист КЦ
void event_write_List_IC(lv_event_t* e)
{
```

```
 lv_event_code_t code_file_or_dir = lv_event_get_code(e);
 tree_value* data_file_or_dir =
(tree_value*)lv_event_get_user_data(e);
 lv_obj_t* obj = lv_event_get_target(e);
 lv_obj_t* btn = obj;
 LV_UNUSED(obj);
 lv_obj_t* cb = lv_obj_get_child(btn, -1);

 EFI_STATUS status;
 IC_FILE_INFO* pFileList = (IC_FILE_INFO*)NULL;
 UINTN FileCount = 0;
 EFI_FILE_PROTOCOL* icdbFile = (EFI_FILE_PROTOCOL*)NULL;
 EFI_FILE_PROTOCOL* hControlFile = (EFI_FILE_PROTOCOL*)NULL;
 IC_FILE_INFO* Ic_Info =
(IC_FILE_INFO*)AllocateZeroPool(sizeof(IC_FILE_INFO));
 CHAR8* pBuffer = NULL;
 UINTN BufferSize = 0;
 Ic_Info->pFileName = (CHAR16*)data_file_or_dir->name;
 CHAR16* szControlFileName = NULL;
 UINTN StrLenFileName;

 if (code_file_or_dir == LV_EVENT_KEY)
 {
      char key = *((char*)lv_event_get_param(e));

      if (LV_KEY_SPACE == key)
      {
           BOOLEAN isDir = FALSE;
           if (data_file_or_dir->type == 0)
           {
                isDir = TRUE;
           }
           if (data_file_or_dir->parent == data_file_or_dir->efs-
>root)
           {
                StrLenFileName = StrLen(Ic_Info->pFileName) + 1 +
1;//слеш + терминальныйноль
                szControlFileName =
(CHAR16*)AllocateZeroPool(StrLenFileName * sizeof(CHAR16));
                if (isDir == FALSE)
                {
                     StrCpyS(szControlFileName, StrLenFileName,
L"\\");
                }
                //StrCpyS(szControlFileName, StrLenFileName,
L"\\");
                StrCatS(szControlFileName, StrLenFileName,
Ic_Info->pFileName);

           }
           else
```

```
            {
                StrLenFileName = StrLen(data_file_or_dir->parent-
>Full_name) + 1 + StrLen(Ic_Info->pFileName) +
1;//терминальныйноль
                szControlFileName =
(CHAR16*)AllocateZeroPool(StrLenFileName * sizeof(CHAR16));

                StrCpyS(szControlFileName, StrLenFileName, L"\\");

                StrCatS(szControlFileName, StrLenFileName,
data_file_or_dir->parent->Full_name);
                StrCatS(szControlFileName, StrLenFileName,
Ic_Info->pFileName);
            }

            if (lv_obj_has_state(cb, LV_STATE_CHECKED)) {
                lv_obj_clear_state(cb, LV_STATE_CHECKED);
                OpenICDB(EFI_FILE_MODE_WRITE | EFI_FILE_MODE_READ
| EFI_FILE_MODE_CREATE, &icdbFile);
                //удалениефайлапоставленногонаКЦ
                //RemoveNode(&g_ElockSettings, icdbFile, Ic_Info,
szControlFileName, curr_volume->fs_index, &pBuffer, &BufferSize);
                RemoveNode(icdbFile, Ic_Info, szControlFileName,
data_file_or_dir->efs->fs_index, &pBuffer, &BufferSize);
            }
            else
            {
                lv_obj_add_state(cb, LV_STATE_CHECKED);
                //запись файла поставленного на КЦ в память
                //status = WriteListIC(&g_ElockSettings, Ic_Info,
szControlFileName, curr_volume->fs_index);
                status = WriteListIC(Ic_Info, szControlFileName,
data_file_or_dir->efs->fs_index, isDir);
            }
            lv_obj_del(list_control);
            Create_list_control(cont_form_down);
            status = ReadAndParseIcdb(&pFileList, &FileCount,
(PIC_FILE_INFO*)NULL, NULL, NULL);

            if (EFI_ERROR(status))
            {
                _DEBUG_PRINT(DEBUG_WARN, "The configuration file
could not be read.");
                return;
            }
            for (UINTN fileIndex = 0; fileIndex < FileCount;
fileIndex++)
            {
                bt_control = lv_list_add_btn(list_control, NULL,
Uint16ToUint8(pFileList[fileIndex].pFileName));
```

```
                lv_obj_add_event_cb(bt_control,
event_handler_control, LV_EVENT_ALL,
pFileList[fileIndex].pFileName);
                count_control++;
           }
      }
 }
}

//обработка событий внутри списка с каталогами и файлами
static void event_handler_file(lv_event_t* e)
{
 lv_event_code_t code_file = lv_event_get_code(e);

 tree_value* data_file = (tree_value*)lv_event_get_user_data(e);
 lv_obj_t* obj = lv_event_get_target(e);
 lv_obj_t* parent = lv_obj_get_parent(obj);

 uint32_t i;

 if (code_file == LV_EVENT_KEY)
 {
      char key = *((char*)lv_event_get_param(e));

      if (LV_KEY_BACKSPACE == key)
      {

           if (data_file->parent == data_file->efs->root)
           {
                //enter_volume_mode_handler();
                if (label_Label)
                {
                     lv_obj_del(label_Label);
                     label_Label = NULL;
                }
                label_Label = lv_label_create(cont_right_panel);
                lv_obj_set_flex_flow(label_Label,
LV_FLEX_FLOW_COLUMN);
                lv_label_set_text(label_Label, "");

                if (list_tom)
                {
                     lv_obj_del(list_tom);
                     list_tom = NULL;
                }

                if (list_file_and_dir)
                {
                     lv_obj_del(list_file_and_dir);
                     list_file_and_dir = NULL;
                }
```

```
                list_file_and_dir = lv_list_create(cont_form_l);

                lv_obj_set_size(list_file_and_dir, 498, 585);
                lv_obj_center(list_file_and_dir);

                enter_volume_mode_handler();
            }
            else
            {
                if (list_file_and_dir)
                {
                    lv_obj_del(list_file_and_dir);
                    list_file_and_dir = NULL;
                }

                list_file_and_dir = lv_list_create(cont_form_l);

                lv_obj_set_size(list_file_and_dir, 498, 585);
                lv_obj_center(list_file_and_dir);
                DirListCycle(data_file->parent->parent);
                lv_event_send(lv_obj_get_child(list_file_and_dir,
0), LV_EVENT_FOCUSED, NULL);
                lv_obj_t* obj_focus =
lv_obj_get_child(list_file_and_dir, 0);
                lv_group_focus_obj(obj_focus);
            }

            LV_LOG_USER("LV_EVENT_KEY,  Key: %d", key);
        }

    }

    if ((code_file == LV_EVENT_FOCUSED))
        {
            lv_obj_t* currentButton = NULL;
            for (i = 0; i < lv_obj_get_child_cnt(parent); i++) {
                lv_obj_t* child = lv_obj_get_child(parent, i);
                if (lv_obj_has_state(child, LV_STATE_FOCUSED)) {
                    currentButton = child;
                    LV_LOG_USER("Current: ");
                    break;
                }
            }

            focused_button = currentButton;

            if (label_Label)
                lv_obj_del(label_Label);

            label_Label = lv_label_create(cont_right_panel);
```

```
            lv_obj_set_flex_flow(label_Label, LV_FLEX_FLOW_COLUMN);
            lv_label_set_text(label_Label, "");

            CHAR16* fileName = (CHAR16*)data_file->name;
            EFI_FILE_INFO* fileInfo = (EFI_FILE_INFO*)NULL;
            UINTN fileInfoSize = 0;

            Create_VolumeAttribute(&label_file_FileName);
            Create_VolumeAttribute(&label_file_Attribute);
            Create_VolumeAttribute(&label_file_CreateTime);
            Create_VolumeAttribute(&label_file_FileSize);
            Create_VolumeAttribute(&label_file_LastAccessTime);
            Create_VolumeAttribute(&label_file_ModificationTime);
            Create_VolumeAttribute(&label_file_PhysicalSize);
            lv_style_set_width(&form_cont_right, 490);
            lv_style_set_height(&form_cont_right, 360);

            EFI_FILE_PROTOCOL* openedFile =
(EFI_FILE_PROTOCOL*)NULL;
            UINTN file_Attribute;
            EFI_TIME file_CreateTime;
            CHAR16* file_FileName;
            UINTN file_FileSize;
            EFI_TIME file_LastAccessTime;
            EFI_TIME file_ModificationTime;
            UINT64 file_PhysicalSize;
            UINT64 file_Size;

            EFI_STATUS status;
            CHAR16* temp_parent_name;
            if (data_file->parent->Full_name)
                 temp_parent_name = data_file->parent->Full_name;
            else
                 temp_parent_name = data_file->parent->name;
            CHAR16* child_file_name =
(CHAR16*)AllocateZeroPool(StrLen(temp_parent_name) *
sizeof(CHAR16) + StrLen(fileName) * sizeof(CHAR16) +
sizeof(CHAR16));

                 if (child_file_name != NULL)
            {
                 memcpy(child_file_name, temp_parent_name,
StrLen(temp_parent_name) * sizeof(CHAR16));
                 memcpy(child_file_name + StrLen(temp_parent_name),
fileName, StrLen(fileName) * sizeof(CHAR16) + sizeof(CHAR16));

                 status = OpenFileByGuid(data_file->efs-
>pVolumeInfoEx->PartitionGuid, child_file_name, &openedFile,
EFI_FILE_MODE_READ);

                 FreePool(child_file_name);
```

```
            }
            LV_LOG_USER("FILE : %s", fileName);
            char volume_info_buffer[500];
            if (status == EFI_SUCCESS)
            {
                GetFileInfo(openedFile, &fileInfo, &fileInfoSize);
                openedFile->Close(openedFile);
                file_Attribute = fileInfo->Attribute;
                file_CreateTime = fileInfo->CreateTime;
                file_FileName = fileInfo->FileName;
                file_FileSize = fileInfo->FileSize;
                file_LastAccessTime = fileInfo->LastAccessTime;
                file_ModificationTime = fileInfo-
>ModificationTime;
                file_PhysicalSize = fileInfo->PhysicalSize;
                file_Size = fileInfo->Size;

                lv_label_set_text_fmt(label_file_FileName,
ConvertStrToUtf8(" Название: %s \n"),
Uint16ToUint8(file_FileName));
                lv_snprintf(volume_info_buffer,
sizeof(volume_info_buffer), " Атрибуты:  %d \n",
file_Attribute);
                lv_label_set_text(label_file_Attribute,
ConvertStrToUtf8(volume_info_buffer));
                lv_snprintf(volume_info_buffer,
sizeof(volume_info_buffer), " Времясоздания: %d.%d.%d \n",
file_CreateTime.Day, file_CreateTime.Month,
file_CreateTime.Year);
                lv_label_set_text(label_file_CreateTime,
ConvertStrToUtf8(volume_info_buffer));
                method(label_file_FileSize, file_FileSize,
"Размерфайла");
                lv_snprintf(volume_info_buffer,
sizeof(volume_info_buffer), " Датапоследнегопрочтенияфайла:
%d.%d.%d \n", file_LastAccessTime.Day,
file_LastAccessTime.Month, file_LastAccessTime.Year);
                lv_label_set_text(label_file_LastAccessTime,
ConvertStrToUtf8(volume_info_buffer));
                lv_snprintf(volume_info_buffer,
sizeof(volume_info_buffer), " Датапоследнегоизмененияфайла:
%d.%d.%d \n", file_ModificationTime.Day,
file_ModificationTime.Month, file_ModificationTime.Year);
                lv_label_set_text(label_file_ModificationTime,
ConvertStrToUtf8(volume_info_buffer));
                method(label_file_PhysicalSize,
file_PhysicalSize, "Размер");
            }
        }
}
```

Приложение В
(Обязательное)

«Программный модуль контроля целостности каталогов и файлов под UEFI»

Презентация

Программный модуль контроля целостности файлов и каталогов под *UEFI*

Автор работы:

студентка гр.21ВВП1 Нефедова Е.Д.

Научный руководитель:

д.т.н. Митрохин М.А.

Преимущества разработки под *UEFI*

Совместимость вычислительных устройств с различным аппаратным обеспечением

UEFI поддерживается на современных платформах, обеспечивая стандартизированное взаимодействие с оборудованием и ОС.

Расширенные возможности безопасности

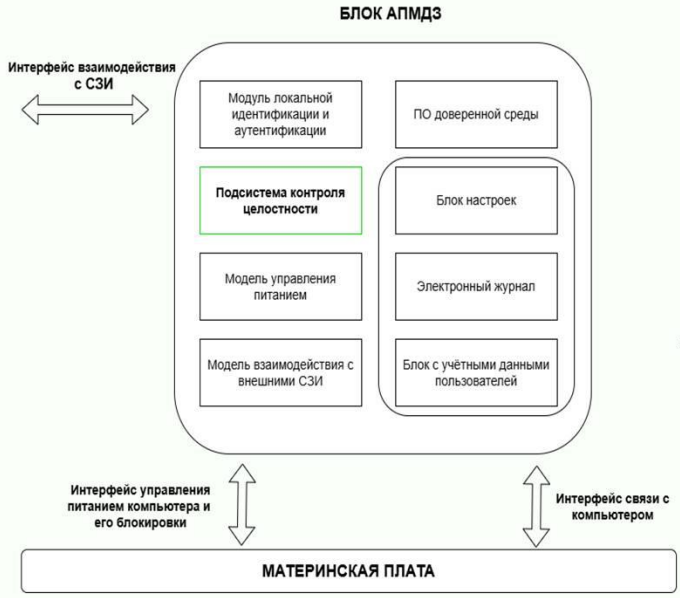
UEFI стандарты активно развиваются в направлении высокой безопасности, предоставляя встроенные механизмы для реализации доверенной загрузки без необходимости интеграции стороннего ПО

Гибкость в настройке и управлении устройствами

Функционирование устройств с программными решениями без необходимости изменения аппаратной части

2

Модуль Доверенной Загрузки (МДЗ) и его функции



3

Почему *LVGL?* Информация о *LVGL* и её преимущества!

Обеспечение визуализации

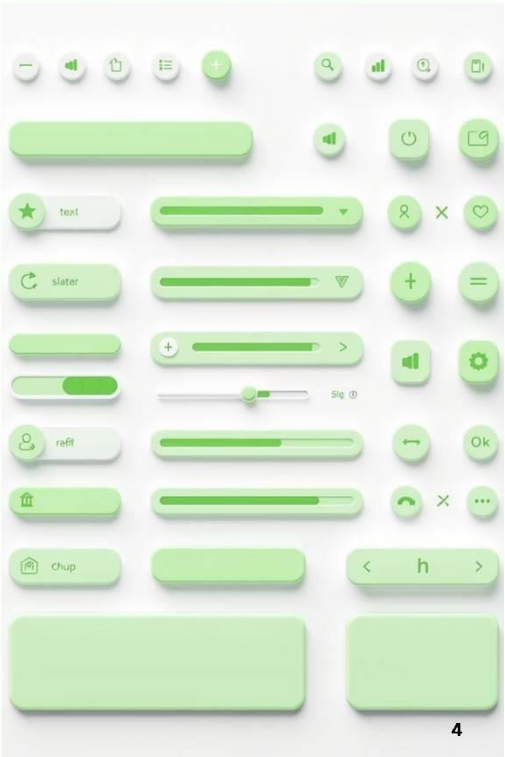
Минимальные требования к ресурсам, что делает её идеальным выбором для устройств с ограниченными вычислительными возможностями

Поддержка ввода пользователя

Инструменты для взаимодействия и настройки параметров через удобные элементы управления.

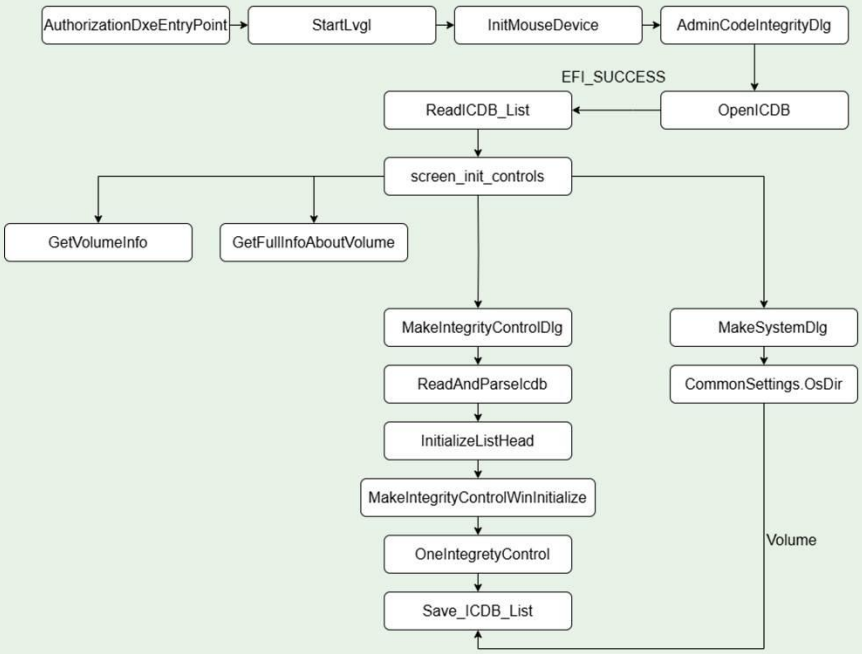
Оптимизация производительности

Лёгкая и быстрая библиотека, адаптирующаяся к ресурсам UEFI-окружения.

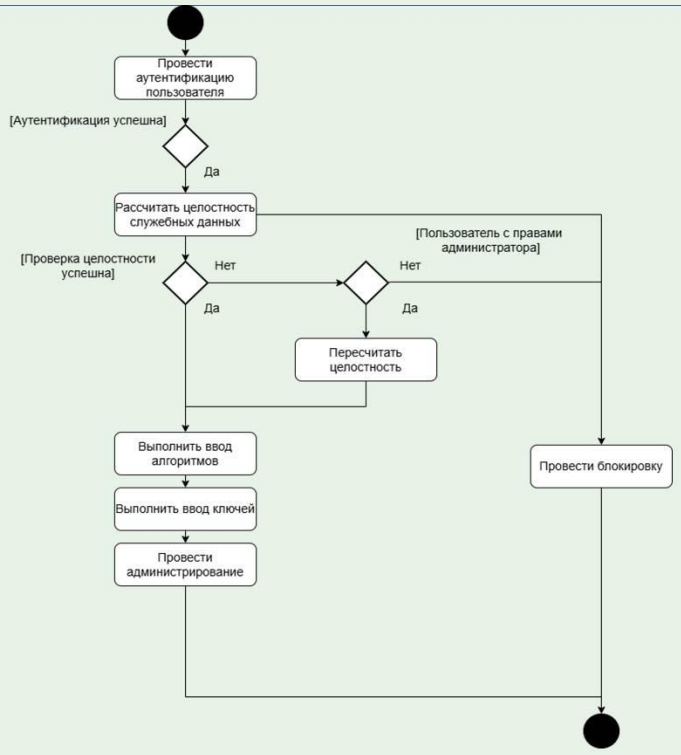


4

Функциональная схема:



5



6

Подсчёт целостности файлов через контрольные суммы *(CRC)*

представление данных в виде двоичного числа, делении его на фиксированный порождающий многочлен и использовании остатка от деления в качестве контрольной суммы

7

Тестирование работы программного модуля:

№	Описание	Результат
1	изменение названия каталога	+
2	изменение названия файла	+
3	удаление каталога	+
4	удаление файла	+
5	изменение содержимого файла	+

8

Спасибо за внимание!