

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Политехнический институт
Факультет приборостроения,
информационных технологий и
электроники

Кафедра
«Информационная безопасность систем
и технологий»

Специальность 10.05.03 –

Информационная безопасность
автоматизированных систем

Специализация –

Защищенные автоматизированные
системы управления

ДИПЛОМНЫЙ ПРОЕКТ

на тему

ЗАЩИТА ОТ НЕСАНКЦИОНИРОВАННОГО ДОСТУПА ИЗДЕЛИЙ
ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ СПЕЦИАЛЬНОГО НАЗНАЧЕНИЯ С
ПРИМЕНЕНИЕМ ДОВЕРЕННЫХ ОТЧУЖДАЕМЫХ НОСИТЕЛЕЙ
ИНФОРМАЦИИ



Студент

Руководитель

Нормоконтролёр

Рецензент зам. ген. директора АО «ПНИЭИ»

Подпись Баркунов

удостоверяю

Мещеряков Владимир Алексеевич

Баркунов А.В.

Фатеев А.Г.

Сомкин С.А.

Начальник бюро кадров

Т.Т. Стальнова

Т.А. Кичасова

Работа допущена к защите (протокол заседания кафедры от 15.06.20 № 13)

Заведующий кафедрой

С.Л. Зефирова

Работа защищена с отметкой хорошо (протокол заседания ГЭК от 25.06.20 № 6)

Секретарь ГЭК

А.П. Иванов

Пенза 2020

Подпись Сомкина удостоверяю

Ведущий инженер

И.П. Орлова



Оглавление

Введение.....	4
1 Обзор методов защиты изделий вычислительной техники от несанкционированного доступа при помощи систем защиты информации и программно-аппаратных комплексов.....	6
1.1 Общие положения.....	6
1.2 Обзор средства защиты информации от несанкционированного доступа «Страж NT 4.0».....	6
1.3 Обзор средства защиты информации от несанкционированного доступа «Secret Net Studio».....	8
1.4 Обзор программно-аппаратного комплекса «Соболь».....	9
1.5 Обзор программно-аппаратного комплекса «Аккорд».....	11
1.6 Обзор электронных идентификаторов и ключей.....	13
1.7 Вывод по результатам анализа существующих средств и методов защиты.....	18
2 Реализация программного обеспечения.....	20
2.1 Общее описание программы.....	20
2.2 Разработка и описание алгоритма программы, отвечающей за формирование графического интерфейса приложения, определение подключенных устройств, операции с меткой и хеш-суммой на накопителе....	21
2.3 Разработка программы, отвечающей за формирование графического интерфейса приложения, определение подключенных устройств, операции с меткой и хеш-суммой на накопителе.....	37
2.4 Разработка и описание алгоритма программы, отвечающей за форматирование.....	64
2.5 Разработка программы, отвечающей за форматирование устройства.....	68
3 Экспериментальная поверка работоспособности программы.....	79

4	Вредные психофизиологические факторы, влияющие на органы зрения специалиста по защите информации. Методы и средства физической культуры, снижающие их воздействие.....	83
4.1	Общие положения.....	83
4.2	Вредные психофизиологические факторы, влияющие на органы зрения специалиста по защите информации.....	83
4.3	Методы и средства физической культуры, снижающие их воздействие.....	86
	Заключение.....	88
	Список используемых источников.....	89
	Приложение А Руководство пользователя.....	91
	Приложение Б Общий алгоритм программы	100
	Приложение В Алгоритм проекта, отвечающего за форматирование.....	102
	Приложение Г Код проекта, отвечающего за формирование графического интерфейса приложения, определение подключенных устройств, получение хеш-суммы с использованием ГОСТ 28147-89 в режиме выработки имитовставки, проверку, чтение, запись хеш-суммы на накопитель.....	103
	Приложение Д Код проекта, отвечающего за форматирование устройства.....	146
	Приложение Е Протокол проверки на оригинальность.....	163

Введение

Устройства обработки и хранения информации на базе современных информационных технологий породили массу способов несанкционированного получения информации или воздействия на нее и, как следствие, способов защиты от этого. Наиболее эффективных результатов в защите информации можно достичь путем создания систем обеспечения информационной безопасности. Эти системы могут включать комплексы мер, создающие преграду вокруг защищаемой информации, состоящую из ряда административно-режимных, правовых, физических, аппаратно-технических, программных и других мер по обнаружению и нейтрализации попыток несанкционированного воздействия на информацию.

Использование программно-аппаратных комплексов и средств защиты информации от несанкционированного доступа с каждым днем все более распространено на рабочих местах в различных организациях. Данные комплексы и средства реализуют множество функций, такие как:

- аутентификация с использованием аппаратных идентификаторов;
- дискреционный и мандатный принцип контроля доступа к ресурсам системы;
- создание замкнутой программной среды пользователя, позволяющей ему запуск только разрешенных приложений;
- регистрация событий безопасности;
- контроль целостности защищаемых ресурсов системы и компонентов системы защиты информации;
- управление носителями информации;
- управление устройствами.

Наиболее распространенным из способов защитить информацию от утечки и несанкционированного распространения в организациях — это использование отчуждаемых носителей информации, которые подписаны

специальным образом и безопасны для использования на рабочих местах. Использование существующих на рынке программно-аппаратных комплексов и средств защиты информации, которые предусматривают возможность создания доверенных носителей, может быть нецелесообразным по причине дополнительных финансовых затрат для организаций при использовании готовых решений, затрат на лицензии, обслуживание, а также существующие комплексы и средства могут содержать функционал, который может быть избыточным при разработке собственных изделий вычислительной техники организацией.

В работе рассмотрены основные модули безопасности и электронные идентификаторы, которые могут быть использованы для авторизации пользователя в сети или на локальном компьютере, защиты электронной переписки, безопасного удаленного доступа к информационным ресурсам, а также хранения данных.

Данный проект посвящен разработке программного обеспечения, реализующего защиту от несанкционированного доступа изделий вычислительной техники специального назначения, путем создания уникальной цифровой последовательности на устройстве, для дальнейшего использования таких доверенных накопителей на вычислительной технике.

1 Обзор методов защиты изделий вычислительной техники от несанкционированного доступа при помощи систем защиты информации и программно-аппаратных комплексов

1.1 Общие положения

Существуют различные системы защиты информации, а также программно-аппаратные комплексы для защиты от несанкционированного доступа, в которых реализованы механизмы идентификации и аутентификации пользователей при помощи отчуждаемых носителей информации. Наиболее распространенными среди них являются:

- СЗИ от НСД «Страж NT»;
- СЗИ от НСД «Secret Net Studio»;
- ПАК «Соболь»;
- ПАК «Аккорд».

1.2 Обзор средства защиты информации от несанкционированного доступа «Страж NT 4.0»

Средство защиты информации от несанкционированного доступа «Страж NT» предназначено для комплексной защиты информационных ресурсов от несанкционированного доступа в соответствии с требованиями законодательства Российской Федерации [1]:

- в автоматизированных системах (АС) до класса защищенности 1Б включительно;
- в государственных информационных системах (ГИС) до 1-го класса защищенности включительно;
- в информационных системах персональных данных (ИСПДн) до 1-го уровня защищенности включительно.

Функции защиты информации:

- двухфакторная аутентификация до загрузки операционной системы (в том числе и для виртуальной среды) с использованием аппаратных идентификаторов:
 - USB-идентификаторов типа Guardant ID, Rutoken, eToken, JaCarta, ESMART Token;
 - смарт-карт Rutoken, eToken, JaCarta, ESMART;
 - iButton;
- дискреционный принцип контроля доступа к ресурсам системы;
- мандатный принцип контроля доступа к ресурсам системы, в том числе возможность одновременной работы с документами разных грифов в одном сеансе (без смены сеанса пользователя);
- создание замкнутой программной среды пользователя, позволяющей ему запуск только разрешенных приложений;
- регистрация событий безопасности, в том числе и действий администратора;
- маркировка выдаваемых на печать документов независимо от печатающего их приложения;
- гарантированная очистка:
 - освобождаемой оперативной памяти;
 - содержимого защищаемых файлов при их удалении;
 - файлов подкачки при завершении работы системы;
- контроль целостности защищаемых ресурсов системы и компонентов системы защиты информации;
- управление пользователями;
- управление носителями информации;
- управление устройствами;

- преобразование информации на отчуждаемых носителях;
- тестирование системы защиты информации.

1.3 Обзор средства защиты информации от несанкционированного доступа «Secret Net Studio»

СЗИ от НСД «Secret Net Studio» является комплексным решением для защиты рабочих станций на уровне данных, приложений, сети, операционной системы и периферийного оборудования. В рамках данного продукта решаются следующие задачи:

- защита от внешних угроз:
 - защита рабочих станций и серверов от вирусов и вредоносных программ;
 - защита от сетевых атак;
 - защита от подделки и перехвата сетевого трафика внутри локальной сети;
 - защищенный обмен данными с удаленными рабочими станциями;
- защита от внутренних угроз:
 - защита информации от несанкционированного доступа;
 - контроль утечек и каналов распространения защищаемой информации;
 - защита от действий инсайдеров;
 - защита от кражи информации при утере носителей.

Использование Secret Net Studio позволяет выполнять требования регуляторов при аттестации (оценке соответствия) информационных систем, в которых обрабатывается конфиденциальная информация, на соответствие различным требованиям российского законодательства (защита

государственных информационных систем до класса К1, защита персональных данных до УЗ1, автоматизированных систем до класса 1Б включительно (гос. тайна с грифом «совершенно секретно»)) [2].

Защита от НСД в Secret Net Studio обеспечивается следующими механизмами:

— защита входа в систему, которая обеспечивает предотвращение доступа посторонних лиц к компьютеру. К механизму защиты входа относятся следующие средства:

- средства для идентификации и аутентификации пользователей;
- средства блокировки компьютера;
- аппаратные средства защиты от загрузки ОС со съемных носителей (интеграция с ПАК «Соболь»);

— процедуры идентификация и аутентификация пользователей, которые выполняются при каждом входе в систему. В системе Secret Net Studio идентификация пользователей осуществляется по одному из трех вариантов:

- по имени (логин и пароль);
- по имени или токену;
- только по токену.

В Secret Net Studio 8.1 поддерживается работа со следующими аппаратными средствами:

- средства идентификации и аутентификации на базе идентификаторов eToken, iKey, Rutoken, JaCarta и ESMART;
- устройство Secret Net Card;
- программно-аппаратный комплекс «Соболь».

1.4 Обзор программно-аппаратного комплекса «Соболь»

Программно-аппаратный комплекс «Соболь» представляет собой программно-аппаратное средство, предназначенное для:

- защиты компьютера от несанкционированного доступа и обеспечения доверенной загрузки;

- создания доверенной программной среды для повышения класса защиты СКЗИ.

Программно-аппаратный комплекс «Соболь» может применяться как устройство, обеспечивающее защиту автономного компьютера, а также рабочей станции или сервера, входящих в состав локальной вычислительной сети [3].

Возможности ПАК «Соболь»:

- идентификация и аутентификация пользователей по электронным идентификаторам, которая обеспечивается до загрузки операционной системы при помощи ключей iButton, iKey, eToken, Rutoken;

- контроль целостности программной и аппаратной среды с помощью электронного замка «Соболь», который контролирует неизменность аппаратной конфигурации компьютера (PCI-устройств, ACPI, SMBIOS и оперативной памяти), BIOS, файлов ОС и реестра Windows, прикладных программ;

- в ПАК «Соболь» обеспечивается запрет загрузки ОС с внешних носителей (USB, FDD, DVD/CD-ROM, LPT, SCSI-порты и др.), что гарантирует загрузку штатной доверенной операционной системы;

- электронный замок «Соболь» осуществляет ведение системного журнала, записи которого хранятся в специальной энергонезависимой памяти;

- сертифицированный аппаратный датчик случайных чисел может быть использован в СКЗИ для генерации надежных ключей шифрования;

- дополнительный модуль сторожевого таймера блокирует доступ к компьютеру при обнаружении попытки отключения электронного замка «Соболь»;

- защита от подбора пароля.

ПАК Соболь имеет сертификаты соответствия ФСТЭК и ФСБ России и может использоваться для [4]:

- обеспечения 1 класса защищенности персональных данных в ИСПДн;

- обеспечения 1 уровня защищенности в государственных и муниципальных информационных системах и в автоматизированных системах управления техническими процессами и производством на критически важных объектах;

- защиты информации, содержащей сведения, составляющие государственную тайну, в автоматизированных системах до класса 1Б включительно.

Согласно сертификатам соответствия ФСБ России и Минобороны России программно-аппаратный комплекс «Соболь»:

- соответствует требованиям к аппаратно-программным модулям доверенной загрузки ЭВМ класса 1Б и может использоваться для защиты информации, содержащей сведения, составляющие государственную тайну, со степенью секретности «совершенно секретно» включительно;

- проверен на отсутствие НДВ по 2 классу;

- подтверждает соответствие руководящим документам по 2 уровню контроля на отсутствие НДВ и возможность использования в автоматизированных системах до класса защищенности 1Б включительно.

1.5 Обзор программно-аппаратного комплекса «Аккорд»

Программно-аппаратный комплекс средств защиты информации «Аккорд» предназначен для разграничения доступа пользователей к рабочим станциям.

Возможности комплекса:

- защита от несанкционированного доступа к ПЭВМ;

- идентификация/аутентификация пользователей до загрузки операционной системы с последующей передачей результатов успешной идентификации/аутентификации в операционную систему;
- аппаратный контроль целостности системных файлов и критичных разделов реестра;
- доверенная загрузка ОС;
- контроль целостности программ и данных, их защита от несанкционированных модификаций;
- создание индивидуальной для каждого пользователя изолированной рабочей программной среды;
- запрет запуска неразрешенных программ;
- разграничение доступа пользователей к массивам данных и программам с помощью дискреционного контроля доступа;
- разграничение доступа пользователей и процессов к массивам данных с помощью мандатного контроля доступа;
- автоматическое ведение протокола регистрируемых событий в энергонезависимой памяти аппаратной части комплекса;
- контроль доступа к USB устройствам.

В качестве аппаратной базы комплекса может использоваться любая из контроллеров «Аккорд» в составе «Аккорд-АМДЗ». СЗИ НСД «Аккорд-АМДЗ» представляет собой аппаратный модуль доверенной загрузки (АМДЗ) для IBM-совместимых ПК — серверов и рабочих станций локальной сети, обеспечивающий защиту устройств и информационных ресурсов от несанкционированного доступа [5].

Комплекс начинает работу сразу после выполнения кода системного BIOS компьютера — до загрузки операционной системы. Контроллеры семейства «Аккорд-АМДЗ» обеспечивают доверенную загрузку ОС, поддерживающих наиболее распространенные файловые системы, включая: FAT12, FAT16, FAT32,

NTFS, HPFS, Ext2, Ext3, Ext4, ReiserFS, FreeBSD UFS/UFS2, Solaris UFS, QNX4, MINIX.

«Аккорд-АМДЗ», как правило, применяется в следующей конфигурации:

- контроллер — представляет собой карту расширения, устанавливаемую в свободный слот материнской платы СBT. Контроллер является универсальным, не требует замены при смене используемого типа операционной системы (ОС);

- съемник информации с контактным устройством, обеспечивающий интерфейс между контроллером комплекса и персональным идентификатором пользователя;

- персональный идентификатор пользователя — специальное устройство, содержащее уникальный признак пользователя, с которым зарегистрированный пользователь входит в систему и который используется системой для определения его прав, а также для регистрации факта доступа и характера выполняемых им работ или предоставляемых ему услуг. В качестве идентификатора пользователя могут использоваться USB-токены, USB-ключи, смарт-карты, устройства ШИПКА (eToken), TouchMemory-идентификаторы (iButton).

1.6 Обзор электронных идентификаторов и ключей

1.6.1 Общие положения.

Представленные на сегодняшний день решения на рынке электронных идентификаторов можно разделить на два вида: токены и смарт-карты. Несмотря на очень сильные различия во внешнем виде, и токены, и смарт-карты являются идентичными устройствами. Основу их составляет специальная микросхема с защищенной памятью. Различие заключается в том, что токены непосредственно подключены к ПК через USB-порт, а смарт-карты требуют дополнительного оборудования — считывателей.

Токен — это компактное устройство в виде USB устройства, которое служит для авторизации пользователя, защиты электронной переписки, безопасного удаленного доступа к информационным ресурсам, а также надежного хранения любых персональных данных. Токен обладает защищенной памятью, где может храниться ценная информация: пароли, цифровые сертификаты, ключи шифрования и электронно-цифровые подписи.

Токен в самом простом варианте применения позволяет заменять небезопасные парольные системы защиты, обеспечивая двухфакторную аутентификацию. Все что должен сделать пользователь для доступа к приложению — подключить токен к USB-порту и набрать PIN-код. Таким образом, осуществляется двухфакторная аутентификация, когда доступ к информации можно получить, только обладая уникальным предметом (токеном) и зная некоторую уникальную комбинацию символов (PIN-код).

Смарт-карты представляют собой пластиковые карты со встроенной микросхемой.

Токены и смарт-карты позволяют решать целый спектр задач, связанных с криптографией, электронной цифровой подписью и аутентификацией. Например, один и тот же токен или смарт-карту сотрудник может использовать для входа в операционную систему, для защищенного обмена информацией с удаленным офисом, для работы с онлайн сервисами, для подписи документов (ЭЦП), для хранения закрытых ключей и сертификатов (технология PKI) и другие задачи.

Далее будут рассмотрены наиболее распространенные на рынке на сегодняшний день решения электронных идентификаторов и ключей.

1.6.2 Обзор электронного идентификатора Guardant ID.

Guardant ID представляет собой устройство, подключаемое к USB-порту компьютера и служащее для хранения идентификационных данных пользователя в энергонезависимой памяти [6].

Guardant ID применяется в качестве персонального идентификатора администратора и пользователей. Функционирование, в том числе в среде операционной системы, обеспечивается компонентами соответствующих средств и систем, в которых данные устройства используются в качестве персональных идентификаторов. Сертифицирован ФСТЭК России на соответствие требованиям РД НДВ – по второму уровню контроля. В соответствии с РД ФСТЭК второй уровень контроля НДВ достаточен для ПО, используемого при защите информации с грифом «СС».

1.6.3 Обзор средства аутентификации Rutoken.

Устройства Rutoken — это аппаратные и программные решения в области аутентификации, защиты информации и электронной подписи. Смарт-карты и токены Rutoken предназначены для использования ключа электронной подписи и ключа проверки электронной подписи. Разработаны на базе криптографических алгоритмов электронной подписи, а также имеют сертификацию ФСБ и ФСТЭК России. Все смарт-карты и токены Rutoken могут дополняться RFID-метками [7].

RFID — способ автоматической идентификации объектов, в котором посредством радиосигналов считываются или записываются данные, хранящиеся в так называемых транспондерах, или RFID-метках.

1.6.4 Обзор устройства iButton.

iButton представляет собой микроэлектронное устройство. Каждое устройство iButton заключено в стальной герметичный цилиндрический корпус и имеет уникальный номер (ID), записываемый в процессе изготовления. Все устройства iButton помещаются в стальной цилиндрический корпус, выполненный по жестким стандартам и выдерживают серьезные механические и температурные нагрузки. Обмен данными с iButton производится через интерфейс 1-Wire. Информация в этом интерфейсе передается по единственному проводнику. Питание iButton получают из этого же проводника, заряжая внутренний конденсатор в моменты, когда на шине нет обмена

данными. Скорость обмена достаточна для обеспечения передачи данных в момент касания контактного устройства. Для подключения iButton к компьютеру компанией выпускаются считыватели (адаптеры), преобразующие сигналы стандартных портов компьютера (RS232, USB) в сигналы 1-Wire [8].

1.6.5 Обзор электронных устройств eToken.

Электронные USB-ключи и смарт-карты eToken представляют собой компактные устройства, предназначенные для обеспечения информационной безопасности. Устройства eToken содержат процессор и модули памяти, функционируют под управлением своей операционной системы, выполняют необходимые прикладные программы и хранят информацию. USB-ключи и смарт-карты eToken базируются на высокозащищенной платформе, разработанной для производства смарт-карт. Поэтому USB-ключи и смарт-карты eToken фактически являются миниатюрным компьютером, обеспечивающим безопасное хранение персональных данных и надежно защищенным от несанкционированного вмешательства [9].

1.6.6 Обзор электронных устройств JaCarta.

Устройства JaCarta представляют собой USB-токены, смарт-карты и модули безопасности для аутентификации, формирования и проверки электронной подписи, шифрования передаваемых данных, а также безопасного хранения объектов и информации пользователя, например пароли, цифровые сертификаты [10].

USB-токены, смарт-карты и модули безопасности JaCarta могут применяться для многофакторной аутентификации, работы с электронной подписью и шифрования передаваемых данных в различных информационных системах и сервисах. В их число входят системы электронного документооборота, электронные торговые площадки, системы дистанционного банковского обслуживания, таможенного декларирования или сдачи отчетности, разнообразные мобильные и веб-приложения, корпоративные

порталы и облачные сервисы. Возможно применение JaCarta в промышленном оборудовании и различных устройствах.

Устройства JaCarta также можно использовать для хранения ключевых контейнеров популярных программных средств криптографической защиты информации (СКЗИ), а также цифровых сертификатов, паролей и лицензий. Для этого предусмотрена защищенная энергонезависимая память, доступная только после ввода PIN-кода.

1.6.7 Обзор электронных устройств ESMART Token.

ESMART Token является современным программно-аппаратным средством аутентификации, формирования электронных подписей и защиты информации. Карта обеспечивает безопасное хранение и использование цифровых сертификатов, ключей шифрования и электронной подписи. Сильной стороной этих смарт-карт является поддержка российских и международных стандартов криптографической защиты данных и программных алгоритмов RSA с длиной ключа до 4096 бит, а также совместимостью с разработчиками СКЗИ. Токен ESMART можно использовать во всех популярных доступных системах [11].

1.6.8 Обзор персональных идентификаторов ikey.

Персональные идентификаторы ikey представляют собой портативные устройства, обеспечивающие аутентификацию пользователя и хранение конфиденциальной информации. Применение iKey повышает уровень защищенности информации на компьютере без дополнительных затрат на приобретение и установку специального программного обеспечения [12].

iKey рекомендуется использовать для решения следующих задач:

- идентификация и строгая двухфакторная аутентификация пользователя;
- аутентификация пользователя при доступе к защищенным корпоративным данным через SSL;

- аутентификация пользователя при доступе к защищенным Web-ресурсам;
- хранение цифровых сертификатов и закрытых ключей пользователя;
- обеспечение безопасного входа в сеть предприятия с любой рабочей станции;
- автоматическая блокировка рабочей станции при извлечении iKey;
- поддержка аутентификации в VPN-клиентах;
- защита электронной почты.

1.7 Вывод по результатам анализа существующих средств и методов защиты

В настоящее время во многих организациях существует политика информационной безопасности, предусматривающая процессы идентификации и аутентификации пользователей. В основном, данное требование выполняется с помощью использования уже существующих программно-аппаратных комплексов и средств защиты информации. В свою очередь, использование существующих на рынке программно-аппаратных комплексов и средств защиты информации может быть нецелесообразным по следующим причинам:

- дополнительные финансовые затраты для организаций при использовании готовых решений, затраты на лицензии, обслуживание;
- существующие комплексы и средства могут содержать функционал, который может быть избыточным при разработке собственных изделий вычислительной техники организацией;
- готовые решения на рынке предусматривают использование дополнительных аппаратных модулей, что может являться недостатком при разработке организацией изделий вычислительной техники специального назначения, так как подобные изделия, в большинстве случаев, производятся без использования сторонних аппаратных модулей. Также, использование дополнительных аппаратных модулей может быть невозможно из-за

ограниченного пространства в корпусе изделия, что в свою очередь может привести к перегреву техники;

— некоторые существующие средства защиты информации от несанкционированного доступа и программно-аппаратные комплексы функционируют только на базе операционной системы Windows.

2 Реализация программного обеспечения

2.1 Общее описание программы

В качестве инструмента реализации программного обеспечения была использована среда разработки Qt Creator 4, а в качестве языка программирования был использован язык C++. Программное обеспечение реализовано в виде исполняемого файла для работы в Alt Linux 8.

Программное обеспечение включает в себя два проекта. Первый проект «disktest» отвечает за формирование графического интерфейса приложения, определение подключенных устройств, формирование хеш-суммы с использованием ГОСТ 28147-89 в режиме выработки имитовставки [13], проверку, чтение, запись хеш-суммы на накопитель.

Второй проект «fdrive» отвечает за форматирование устройства и создание с помощью встроенной утилиты Linux «msdosfs» файловой системы на накопителе.

Графический интерфейс программного обеспечения содержит в себе таблицу и кнопки «Обновить», «Метка», «Выход».

Таблица содержит в себе столбцы «Модель», «Устройство», «Серийный номер», «Размер(МБ)», «Файловая система», «Метка». Строки таблицы заполняются в процессе работы программы. Графический интерфейс представлен на рисунке 1.

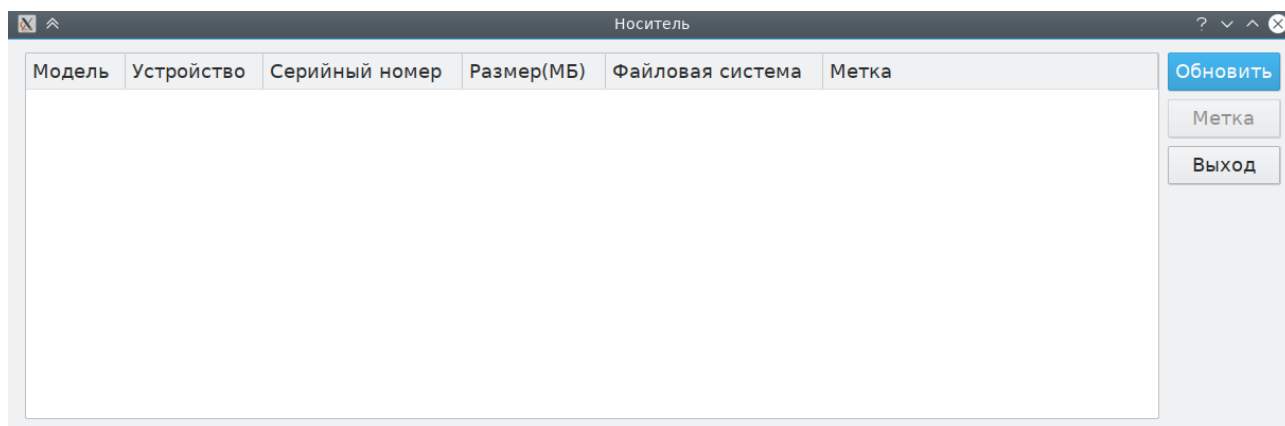


Рисунок 1 — Графический интерфейс программы

Программное обеспечение работает с файловыми системами FAT12, FAT16, FAT32, NTFS, ext2, ext3, ext4. Не допускается удалять предложенную в окне метку носителя «MyFlash», однако она может быть дополнена цифрами в конце, например, метка носителя может быть «MyFlash1». Максимальная длина метки – 11 символов, что позволяет создать десять тысяч уникальных меток.

По разработанной программе было написано руководство пользователя, которое приведено в приложении А.

Общий алгоритм программы представлен в приложении Б. Алгоритм проекта, отвечающего за форматирование накопителя, представлен в приложении В. Код проекта `disktest`, отвечающего за формирование графического интерфейса приложения, определение подключенных устройств, формирование хеш-суммы с использованием ГОСТ 28147-89 в режиме выработки имитовставки [13], проверку, чтение, запись хеш-суммы на накопитель, представлен в приложении Г. Код проекта `fdrive`, отвечающего за форматирование устройства и создание с помощью встроенной утилиты Linux «`msdosfs`» файловой системы на накопителе, представлен в приложении Д.

2.2 Разработка и описание алгоритма программы, отвечающей за формирование графического интерфейса приложения, определение подключенных устройств, операции с меткой и хеш-суммой на накопителе

Был разработан алгоритм проекта «`disktest`», отвечающего за формирование графического интерфейса приложения, определение подключенных устройств, формирование хеш-суммы с использованием ГОСТ 28147-89 в режиме имитовставки [13], проверку, чтение, запись хеш-суммы на накопитель. Проект включает в себя следующие основные функции:

- функцию обновления списка устройств «`reloadLocal`»;
- функцию формирования списка устройств «`mkPartitionList`»;
- функцию получения информации об устройстве «`getDevInfo`»;

- функцию чтения хеш-суммы на накопителе «readHash»;
- функцию получения метки носителя «getLabel»;
- функцию заполнения списка устройств в графическом интерфейсе «fillTree»;
- функцию проверки хеш-суммы «checkSerialHash»;
- функцию вычисления хеш-суммы «passwordhash»;
- функция, реализующая ГОСТ 28147-89 в режиме выработки имитовставки «giz_3»;
- функцию запуска форматирования накопителя «formatting_local_drive»;
- функцию шифрования «hashSerialNumber»;
- функцию ГСЧ «VYR_GSCH»;
- функцию для получения символа из числовой последовательности «mksalt»;
- функцию записи хеш-суммы на накопитель «write_hash».

При запуске программы первоначально доступны только кнопки «Обновить» и «Выход». По нажатию кнопки «Обновить» происходит очистка строк таблицы, инициализация буферов для последующего чтения и записи, и происходит вызов функции обновления списка устройств «reloadLocal». В свою очередь, функция «reloadLocal» вызывает функцию формирования списка устройств «mkPartitionList», по результату работы которой заполняет таблицу новым списком накопителей, которые удалось определить программе. Функция «mkPartitionList» производит поочередную проверку всех устройств, изменяя в пути к устройству последнюю букву английского алфавита таким образом, что первый проверяемый путь будет «/dev/sda», а последний «/dev/sdz». Данный способ необходим, чтобы все устройства, включая тех, которые еще не были инициализированы операционной системой, были определены. Затем

проводится чтение системного файла операционной системы Linux по пути «/proc/partition». В данном файле перечислены все устройства, подключенные в данный момент в системе. Из этого файла считываются все необходимые устройства, исключая ненужные, например, модули оперативной памяти «ram». В процессе работы данной функции вызывается следующая функция – «getDevInfo», позволяющая получить необходимую информацию об устройстве, в том числе считывает хеш-сумму с устройства, при помощи функции «readHash», и метку, при помощи функции «getLabel», которая читает определенный участок памяти, в зависимости от файловой системы устройства. Таким образом, мы получаем все необходимые данные для таблицы, а именно: модель устройства, путь обращения к устройству, серийный номер устройства и результат проверки хэша, объем накопителя в мегабайтах, данные о файловой системе, метку носителя. Алгоритм работы программы по нажатию кнопки «Обновить» представлен на рисунке 2.

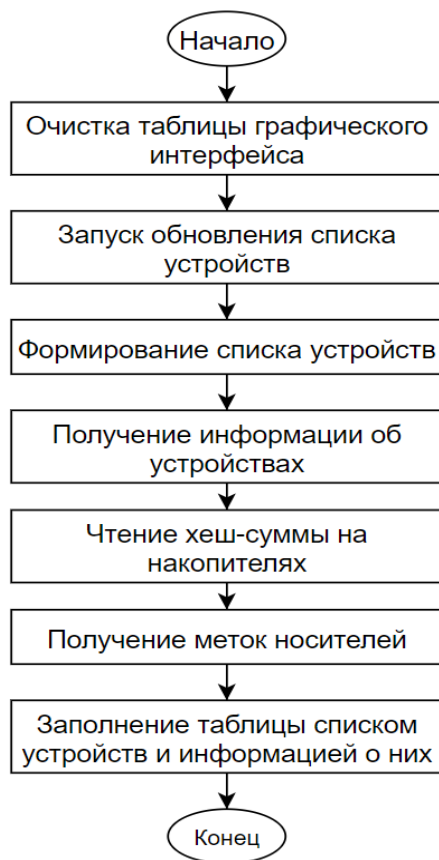


Рисунок 2 — Алгоритм работы программы по нажатию кнопки «Обновить»

После нажатия кнопки «Обновить» и заполнения таблицы списком устройств, становится доступна кнопка «Метка». По нажатию данной кнопки программа сначала выдает предупреждение о том, что произойдет смена метки на устройстве, а затем выдает предупреждение, что все данные на носителе будут уничтожены. Если пользователь нажмет кнопку «ОК», а затем «Продолжить» в появившемся окне, то произойдет вызов функции форматирования «formatting_local_drive», по завершению которой программа выполнит аналогичные действия, что и по нажатию кнопки «Обновить», работа которой описана ранее. В функции «formatting_local_drive» указан путь ко второму проекту «fdrive», отвечающему за форматирование устройства. Перед запуском работы «fdrive» выполняется вызов функции шифрования «hashSerialNumber», в которой генерируется «соль» при помощи функции-генератора случайных чисел «VYR_GSCH» и функции для получения символа из числового значения «mksalt». Далее в функции «hashSerialNumber» производится вызов функции вычисления хеш-суммы «passwordhash». Функция «passwordhash» осуществляет вычисление хеш-суммы, используя функцию «giz_3», которая представляет собой реализацию ГОСТ 28147-89 в режиме выработки имитовставки [13]. После выполнения функции «hashSerialNumber» производится запуск проекта «fdrive», который будет описан далее в п.2.4 данной работы. В случае удачного форматирования устройства, запускается функция записи на накопитель ранее полученной хеш-суммы «write_hash». По завершению работы выдается сообщение о результате. При нажатии кнопки «Выход» произойдет завершение работы программы и ее закрытие. Алгоритм работы программы по нажатию кнопки «Метка» представлен на рисунке 3.

На рисунке 4 представлен алгоритм функции обновления списка устройств «reloadLocal».

На рисунке 5 представлен алгоритм функции формирования списка устройств «mkPartitionList».

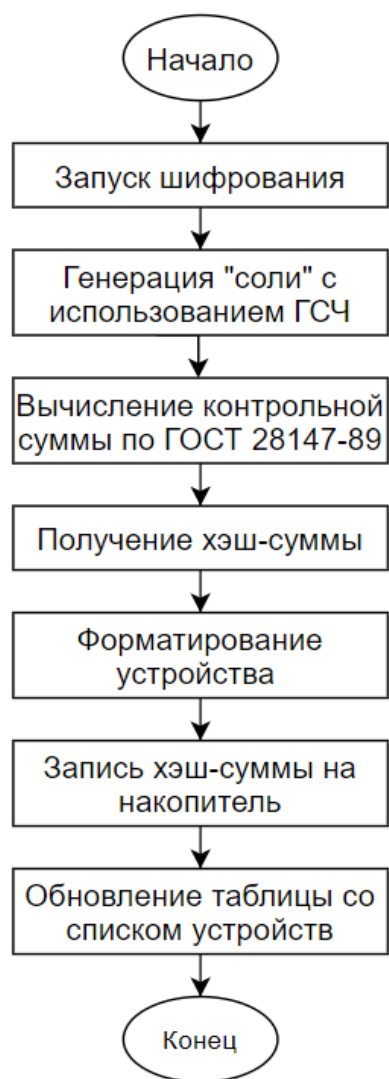


Рисунок 3 — Алгоритм работы программы по нажатию кнопки «Метка»

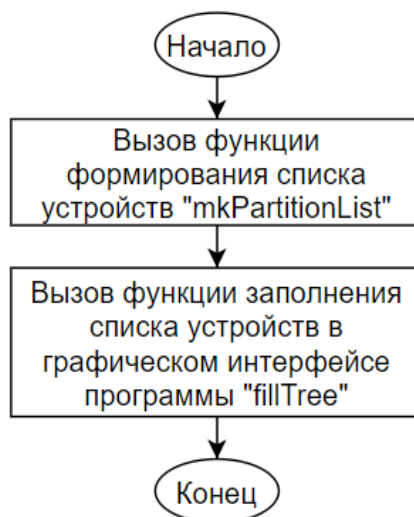


Рисунок 4 — Алгоритм функции обновления списка устройств «reloadLocal»

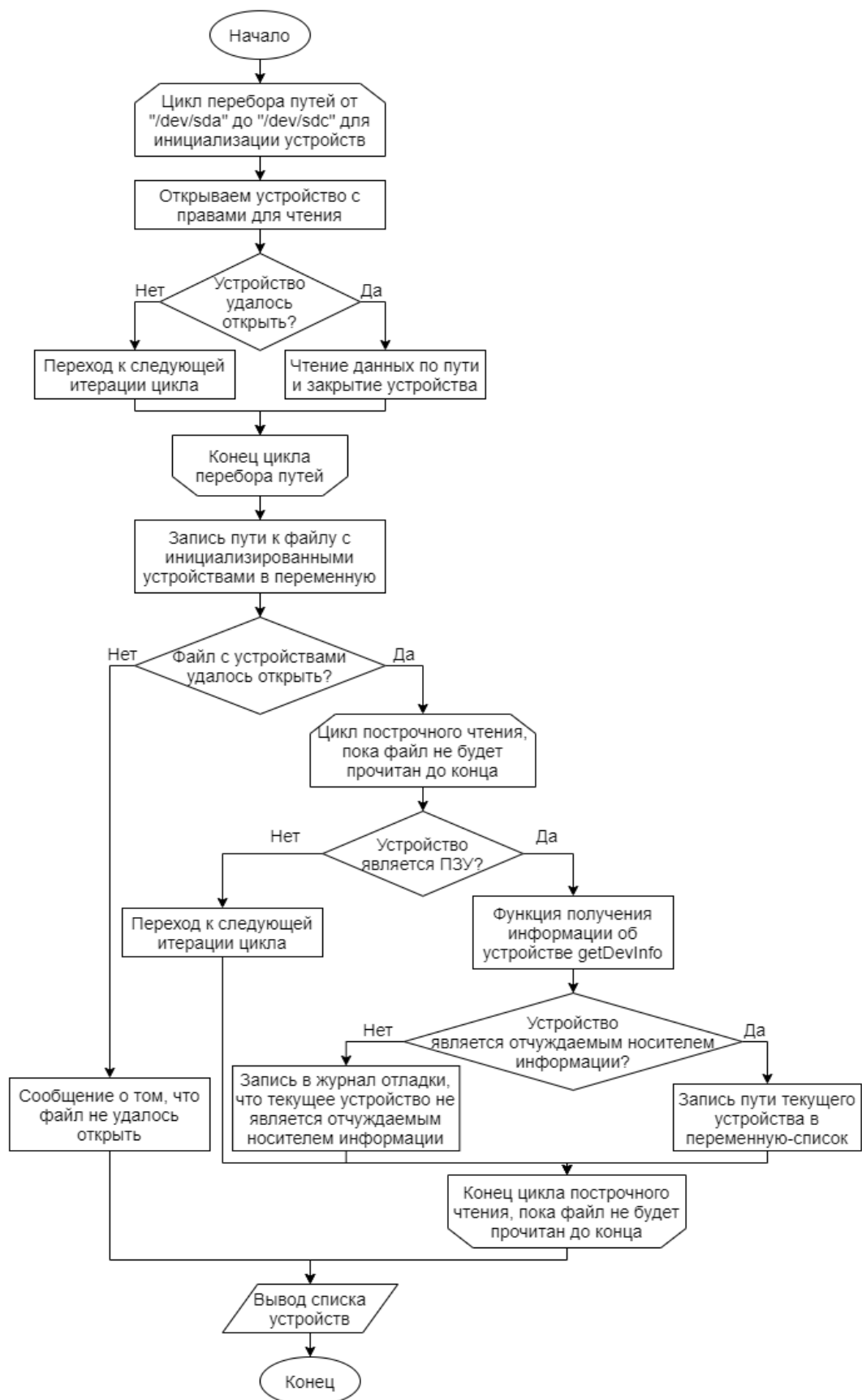


Рисунок 5 — Алгоритм функции формирования списка устройств «mkPartitionList»

На рисунке 6 представлен алгоритм функции получения информации об устройстве «getDevInfo».

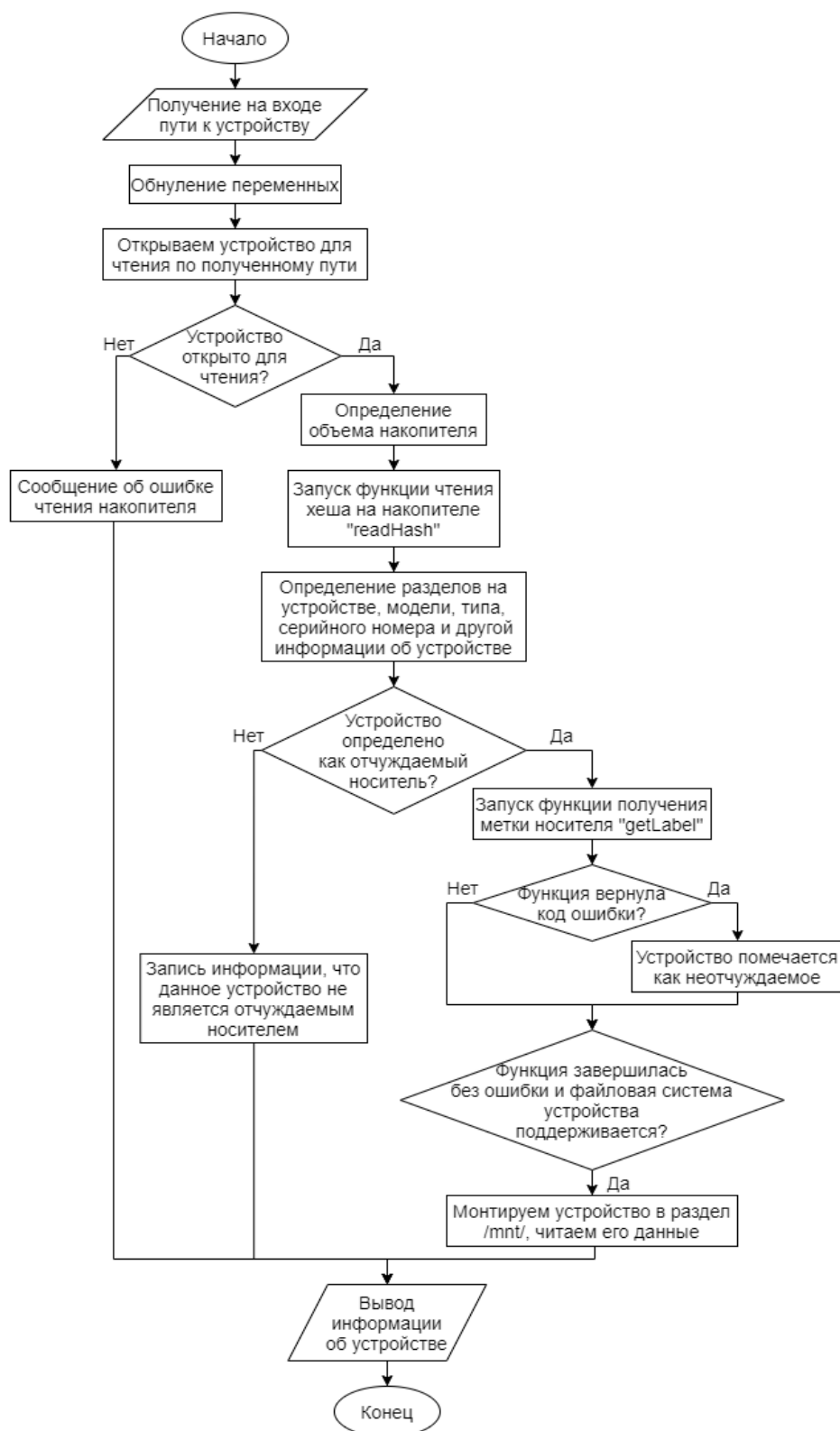


Рисунок 6 — Алгоритм функции получения информации об устройстве «getDevInfo»

На рисунке 7 представлен алгоритм функции чтения хеш-суммы «ReadHash».

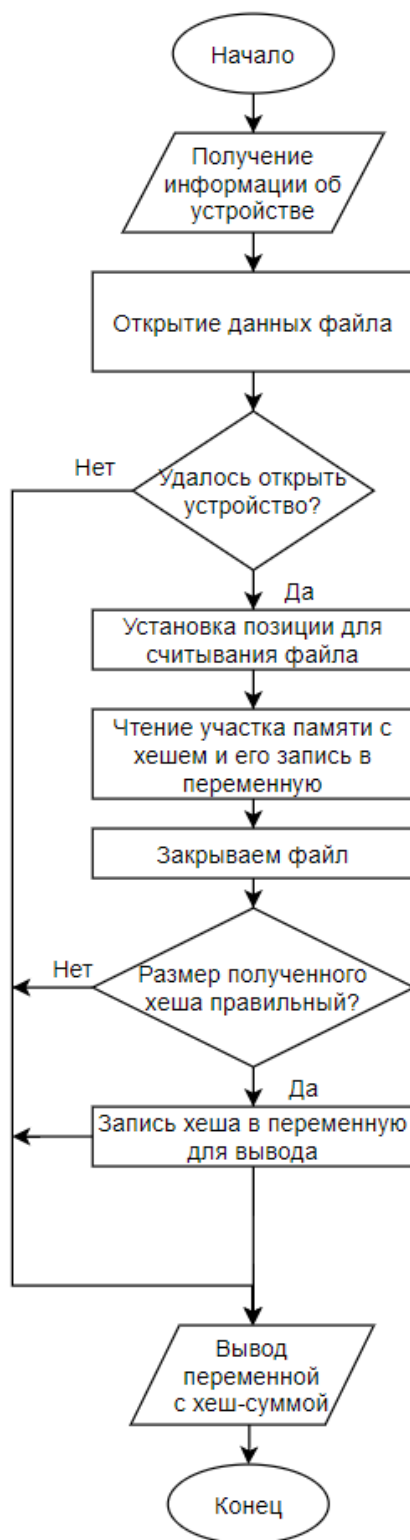


Рисунок 7 — Алгоритм функции функции чтения хеш-суммы «ReadHash»

На рисунке 8 представлен алгоритм функции получения метки носителя «getLabel».

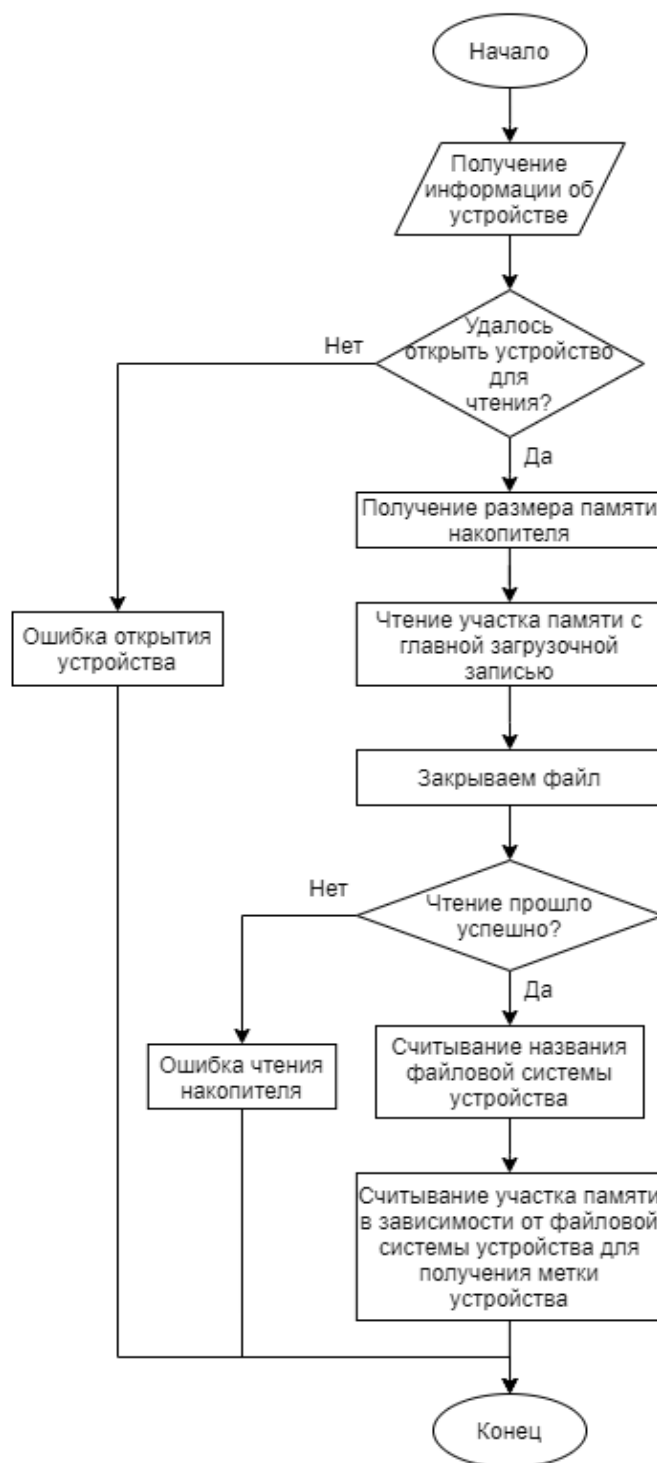


Рисунок 8 — Алгоритм функции получения метки носителя «getLabel»

На рисунке 9 представлен алгоритм функции заполнения списка устройств в графическом интерфейсе «fillTree».

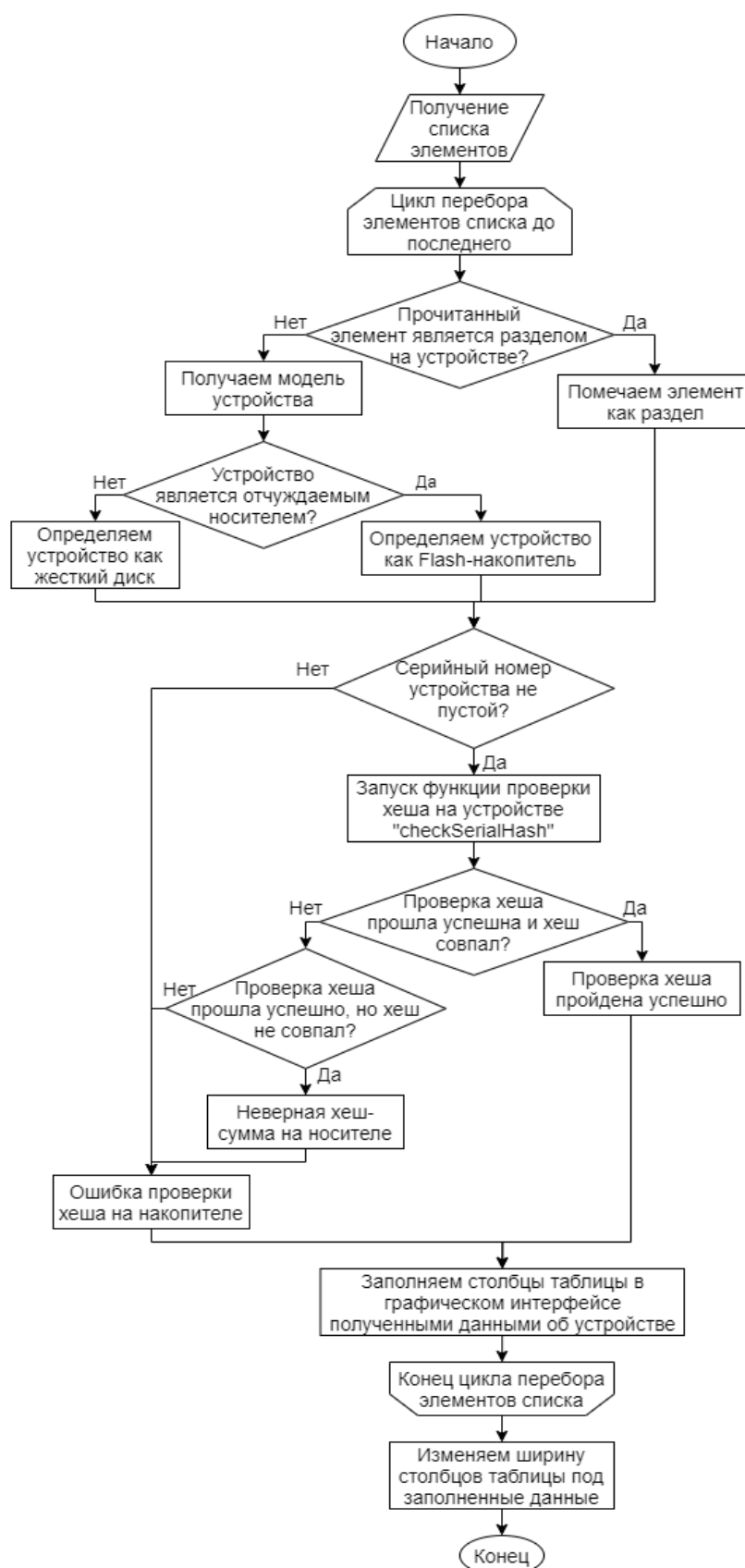


Рисунок 9 — Алгоритм функции заполнения списка устройств в графическом интерфейсе «fillTree»

На рисунке 10 представлен алгоритм функции проверки хеш-суммы «checkSerialhash».

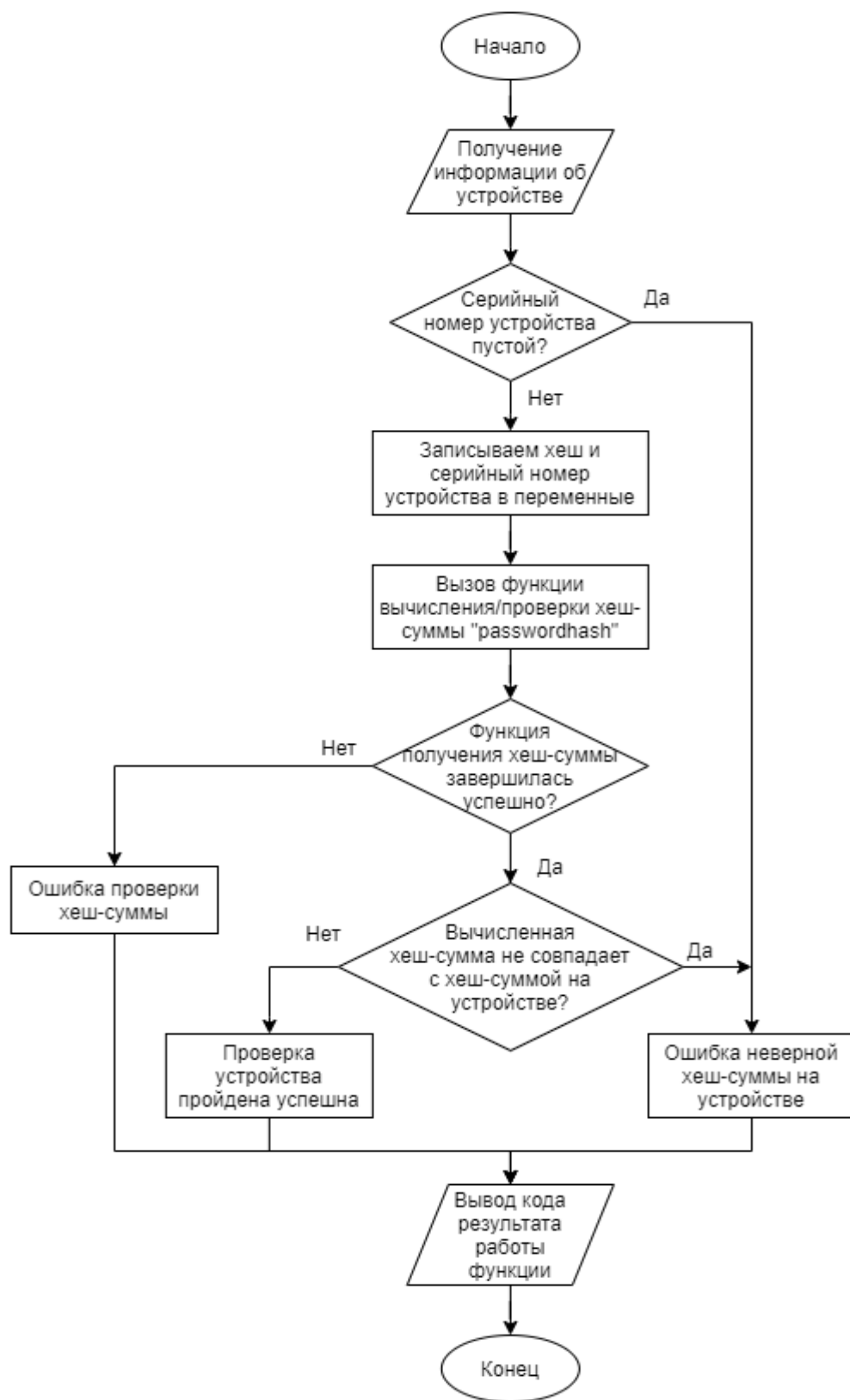


Рисунок 10 — Алгоритм функции проверки хеш-суммы «checkSerialhash»

На рисунке 11 представлен алгоритм функции вычисления хеш-суммы «passwordhash».

На рисунке 12 представлен алгоритм функции реализующей ГОСТ 28147-89 в режиме выработки имитовставки «giz_3».

На рисунке 13 представлен алгоритм функции запуска форматирования накопителя «formatting_local_drive».

На рисунке 14 представлен алгоритм функции шифрования «hashSerialNumber».



Рисунок 11 — Алгоритм функции вычисления хеш-суммы «passwordhash»

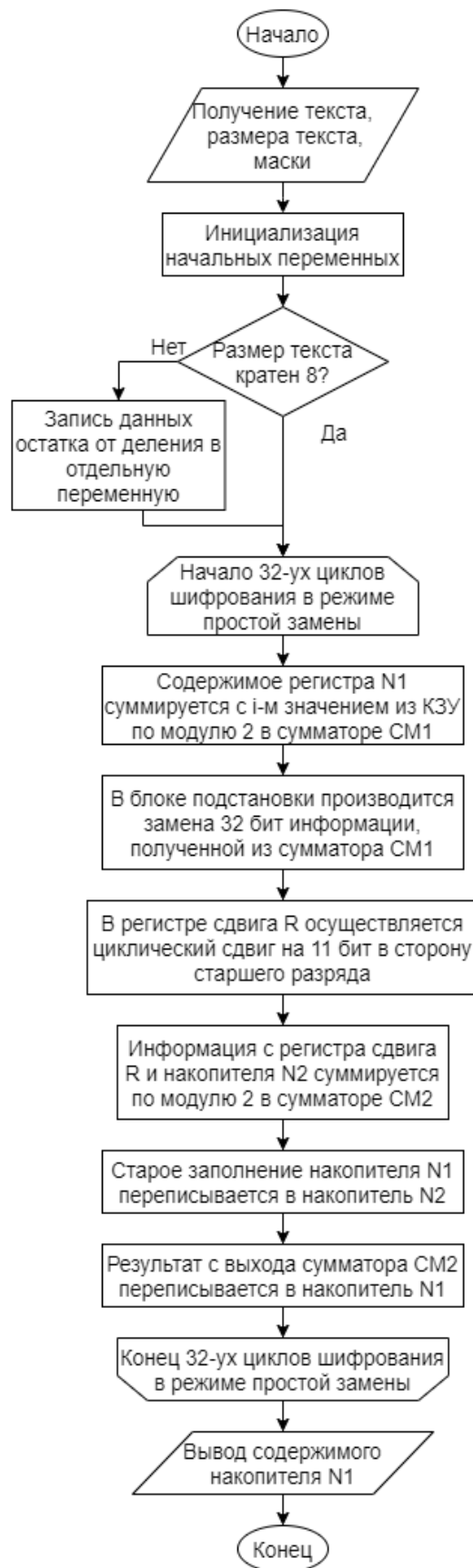


Рисунок 12 — Алгоритм функции реализующей ГОСТ 28147-89 в режиме
выработки имитовставки «giz_3»

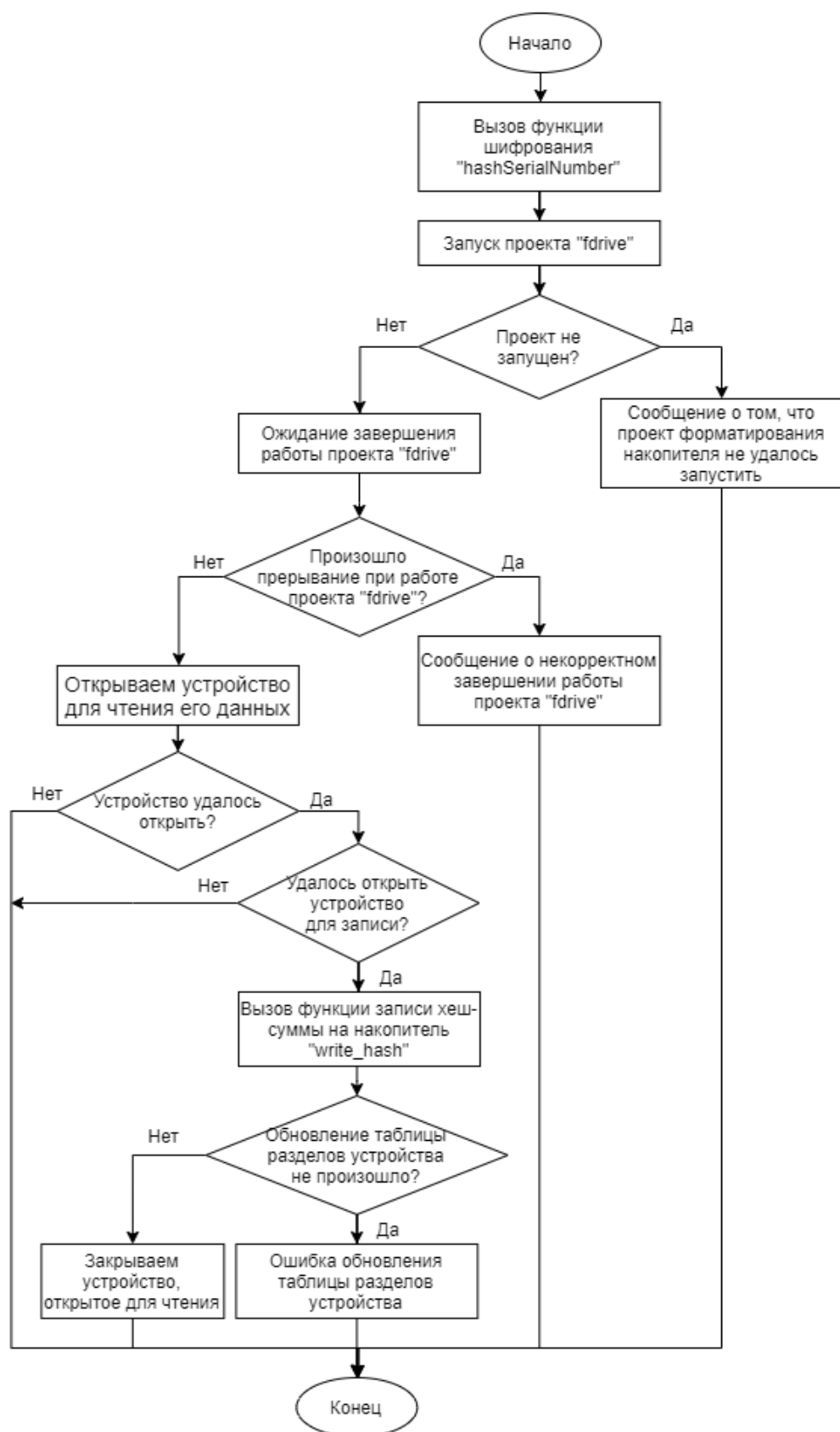


Рисунок 13 — Алгоритм функции запуска форматирования накопителя
«formatting_local_drive»

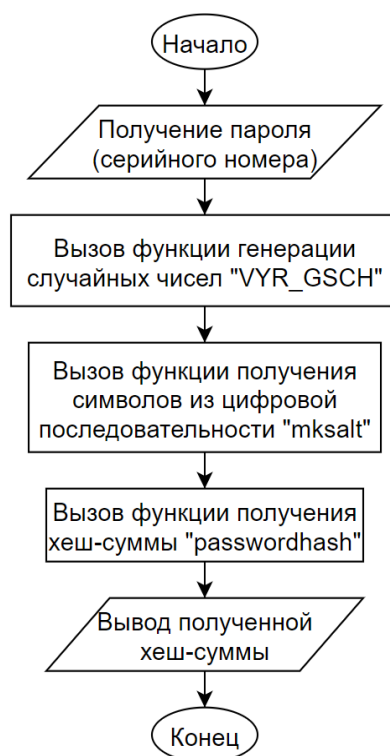


Рисунок 14 — Алгоритм функции шифрования «hashSerialNumber»

На рисунке 15 представлен алгоритм функции реализующей ГСЧ «VYR_GSCH».

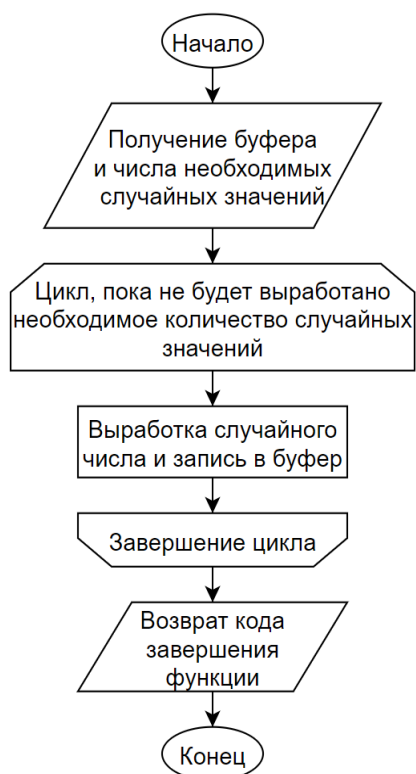


Рисунок 15 — Алгоритм функции реализующей ГСЧ «VYR_GSCH»

На рисунке 16 представлен алгоритм функции получения символа из числовой последовательности «mksalt».

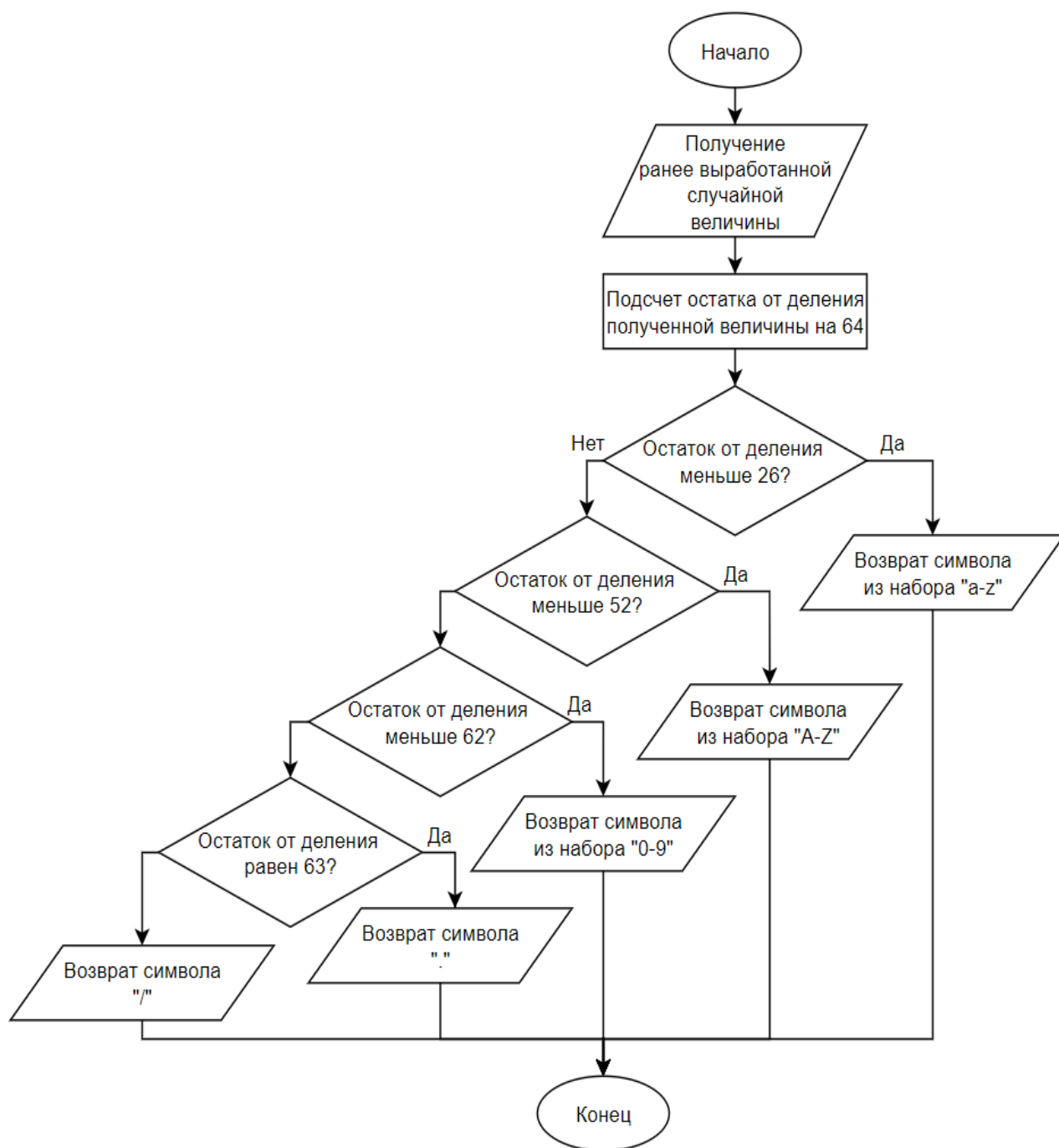


Рисунок 16 — Алгоритм функции получения символа из числовой последовательности «mksalt»

На рисунке 17 представлен алгоритм функции записи хеш-суммы на накопитель «write_hash».

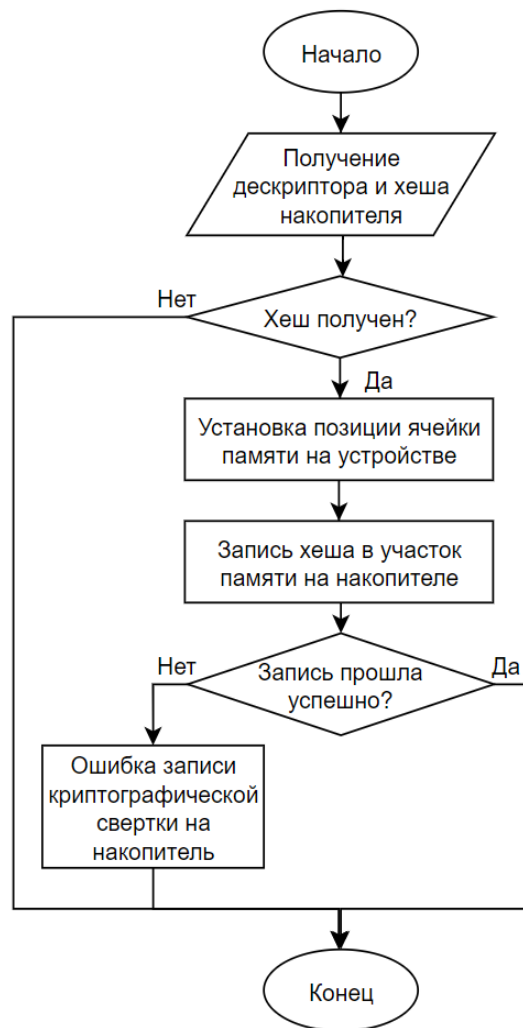


Рисунок 17 — Алгоритм функции записи хеш-суммы на накопитель
«write_hash»

2.3 Разработка программы, отвечающей за формирование графического интерфейса приложения, определение подключенных устройств, операции с меткой и хеш-суммой на накопителе

2.3.1 Описание начальных переменных.

Был рассмотрен заголовочный файл `disktest.h`. В начале файла происходит объявление структуры `TstDevInfo`:

```
typedef struct
{
    QByteArray fileDev;
    QByteArray mountPoint;
```

```

    QByteArray vendor;
    QByteArray model;
    QByteArray label;
    QByteArray serialNumber;
    QByteArray hash;
    off64_t devSize;
    int fs_type;
    int removable;
    int dev_type;
    bool reg;
} TstDevInfo;

```

В данной структуре содержатся переменные для хранения определенной информации об устройстве, а именно:

- fileDev – имя файла устройства;
- mountPoint – точка монтирования;
- vendor – производитель;
- model – модель;
- label – метка;
- serialNumber – серийный номер;
- hash – хеш-сумма;
- devSize – объем памяти устройства;
- fs_type – тип файловой системы;
- removable – признак съемного носителя;
- dev_type – тип устройства;
- reg – признак регистрации носителя.

Затем происходит объявление структуры параметров файловой системы NTFS ntfs_param:

```

struct ntfs_param
{

```

```

unsigned long sec_size;
unsigned long clust_size;
unsigned long file_size;
unsigned long mft_addr;
unsigned long mft_mirror_addr;
unsigned long addr_upseq;
int size_upmass;
unsigned long attr_offset;
int label_offset;
unsigned long label_size;
QString label;
};

```

В данной структуре содержатся переменные для хранения информации об устройстве с файловой системой NTFS, а именно:

- `sec_size` — размер сектора в байтах;
- `clust_size` — размер кластера в байтах;
- `file_size` — размер файла в байтах;
- `mft_addr` — адрес записи mft (файловой таблицы тома);
- `mft_mirror_addr` — адрес копии записи mft;
- `addr_upseq` — адрес последовательности обновления;
- `size_upmass` — размер массива обновления;
- `attr_offset` — смещение атрибута;
- `label_offset` — смещение метки;
- `label_size` — размер метки;
- `label` — метка.

Затем происходит объявление структуры из стандартного заголовочного файла `ext2.h` операционной системы Linux `ext2_super_block`:

```

struct ext2_super_block
{

```

```

char s_dummy0[56];
unsigned char s_magic[2];
char s_dummy1[34];
unsigned int s_feature_compat;
char s_dummy3[24];
char s_volume_name[VOLNAMSZ];
char s_last_mounted[64];
char s_dummy2[824];
}__attribute__((packed));

```

Данная структура необходима для работы с файловой системой системного ядра Linux ext2.

Далее происходит объявление переменной, содержащей названия файловых систем, для последующего заполнения с помощью их одного из столбцов таблицы графического интерфейса в процессе работы программы, и перечисление файловых систем, используемых в программе:

```

static const char * const fs_types[] = {"msdos", "msdos", "vfat",
"ext2", "ext3", "ext4", "ntfs", "отсутствует"};
enum
{
    FAT12 = 0,
    FAT16,
    FAT32,
    EXT2,
    EXT3,
    EXT4,
    NTFS,
    NOT_SUPPORTED
};

```

Далее в заголовочном файле приведены функции, алгоритмы которых были описаны ранее.

2.3.2 Функция обновления списка устройств.

Код функции обновления списка устройств reloadLocal приведен далее:

```

void disktest::reloadLocal()
{
    QList<TStDevInfo> partList = mkPartitionList(partList);

    fillTree (ui->treeWidget->invisibleRootItem(), partList);
}

```

В данной функции происходит заполнение переменной-листа partlist при помощи данных, полученных в ходе работы функции формирования списка устройств mkPartitionList, код которой будет рассмотрен далее. Затем происходит заполнение таблицы графического интерфейса при помощи функции fillTree, код которой будет описан далее.

2.3.3 Функция формирования списка устройства.

Сначала в функции mkPartitionList выполняются следующие команды:

```

{
    QList<TStDevInfo> dev_list;
    QString m_dev = "", t_str = "";
    // Пробежимся по всем файлам устройств для того, чтобы в
системе определились флешки
    char ud_sd[]="/dev/sda";
    char rb, rb_len;
    for(ud_sd[7]='a'; ud_sd[7]!='z'; ud_sd[7]++)
    {
        int ffd = ::open(ud_sd, O_RDONLY, 0444);
        if( ffd != -1 )
        {
            rb_len=::read(ffd, &rb, 1);
            ::close(ffd);
        }
    }
}...

```

В приведенном блоке выполняется цикл перебора путей обращения к устройствам в системе, начиная от «/dev/sda», и заканчивая путем «/dev/sdz». В цикле происходит попытка чтения устройства по пути, зависящего от текущей

итерации цикла. Данное действие необходимо, чтобы в системе определились все устройства подключенные в данный момент.

Далее идет блок кода:

```
...   QFile f_dev("/proc/partitions");
      if(f_dev.open(QIODevice::ReadOnly))
      {
          QTextStream tStream(&f_dev);

          // пропускаем первые две строчки
          m_dev = tStream.readLine();
          m_dev = tStream.readLine();

          while(!tStream.atEnd())
          {
              m_dev = tStream.readLine();

              t_str = m_dev.section(' ', -1);

              if(      t_str.contains("loop")
                    || t_str.contains("sr")
                    || t_str.contains("etherd")
                    || t_str.contains("mtdblock")
                    || t_str.contains("mmcblk0")
                    || t_str.contains("ram")
              )
                  continue;...
```

В приведенном блоке открывается системный файл, расположенный по пути "/proc/partitions", в котором приведен список всех модулей, которые на данный момент функционируют на вычислительном устройстве. Происходит чтение этого файла и исключение всех ненужных для текущей программы модулей, например, из списка удаляются модули оперативной памяти «ram».

Далее идет блок кода:

```

...    if( !QFile::exists( QByteArray("/dev/").append(t_str).data() )
    ){

        unsigned int major = m_dev.section( ' ', 0, 0,
QString::SectionSkipEmpty ).toInt();

        unsigned int minor = m_dev.section( ' ', 1, 1,
QString::SectionSkipEmpty ).toInt();

        if( mknod(QByteArray("/dev/").append(t_str).data(), 0600 |
S_IFBLK, makedev(major, minor)) )

            {

                syslog(LOG_ERR, "Ошибка, %s:%d, Не удалось
создать устройство %s: %m", __FILE__, __LINE__,
QByteArray("/dev/").append(t_str).data());

                QMessageBox::information(0, "", "Не удалось
создать устройство " + QByteArray("/dev/").append(t_str));

            }

        }...

```

В приведенном блоке обрабатывается ошибка, которая может возникнуть, если создать устройство по текущему пути не удалось.

Затем идет блок кода:

```

...    TStDevInfo stDevInfo =
getDevInfo(QByteArray("/dev/").append(t_str).data(), true); //
здесь выдает ошибку об устройстве

        if( stDevInfo.removable != -1 ){

            dev_list += stDevInfo;

        } else {

            syslog(LOG_INFO, "%s removable == -1",
t_str.data());

        }

    }

    f_dev.close();...

```

В приведенном блоке вызывается функция получения информации об устройстве `getDevInfo`, описание которой будет приведено далее в работе. В процессе выполнения команд данного блока заполняется структура данных с

информацией об устройстве. Затем, если устройство было определено как отчуждаемый носитель информации, список устройств дополняется. Далее ранее открытое для чтения устройство закрывается.

Далее по функции в случае, если не удалось открыть устройство, выдается ошибка. На выход функция выдает список устройств.

2.3.4 Функция получения информации об устройстве.

Сначала в функции `getDevInfo` выполняются следующие команды:

```
TStDevInfo disktest::getDevInfo(const char *dev, bool partition)
{
    TStDevInfo stDevInfo;
    stDevInfo.fs_type = NOT_SUPPORTED;
    stDevInfo.fileDev = dev;
    stDevInfo.removable = 0;
    stDevInfo.dev_type = 0;
    int fd = ::open(stDevInfo.fileDev.data(), O_RDONLY |
O_NONBLOCK);...
```

В приведенном блоке происходит обнуление некоторых переменных структуры с информацией о устройстве, затем происходит попытка чтения данных устройства. Далее идет блок кода:

```
...    if(fd >= 0)
        {
            stDevInfo.devSize = ::lseek64(fd, 0, SEEK_END);
            ::close(fd);
            stDevInfo.hash = readHash(stDevInfo); ...
```

В данном блоке проверяется было ли открыто для чтения устройство. Если данные удалось прочитать, то происходит определение объема накопителя и запись полученного размера в переменную структуры. Затем происходит вызов функции чтения хеш-суммы носителя `readHash`, которая будет описана далее в работе. Далее идет блок кода:

```
...        bool ok = false;
            if(partition)
```

```

        ok = (stDevInfo.fileDev.right(1).toInt() == 0
&& !getSCSIDevInfo(&stDevInfo, partition)) ||
(stDevInfo.fileDev.right(1).toInt() != 0);
        if(ok)
        {
            int ret = getLabel(stDevInfo, partition);
            if(ret)
                stDevInfo.removable = -1;
            if(!ret&& stDevInfo.fs_type != NOT_SUPPORTED)
            {
                stDevInfo.mountPoint = stDevInfo.fileDev + "/";
                stDevInfo.mountPoint.replace("dev", "mnt");
                QDir dir(stDevInfo.mountPoint.data());
                if(dir.exists())
                    dir.rmpath(stDevInfo.mountPoint.data());
            }
        }
        else
            stDevInfo.removable = -1; ...

```

В приведенном блоке происходит чтение разделов данных на носителе. Если разделы успешно прочитаны, то вызывается функция получения метки носителя `getLabel`, которая будет описана далее в работе. Если метка не получена, устройство помечается, как неотчуждаемый носитель информации. Если метка устройства получена успешно и файловая система на накопителе поддерживается текущей программой, устройство монтируется в системный каталог «`/mnt/`».

Далее по функции происходит обработка исключения, если накопитель не удалось прочитать. Функция возвращает структуру с информацией об устройстве.

2.3.5 Функция чтения хеш-суммы накопителя.

Сначала в функции `readHash` выполняются следующие команды:

```

QByteArray disktest::readHash(TStDevInfo devInfo)
{
    QByteArray ret;
    int file_fd = ::open(devInfo.fileDev.data(), O_RDONLY);
    char hash_buff[HASH_SIZE] = {0,};
    if(file_fd < 0)
        return ret; ...

```

В приведенном блоке объявляется массив данных, открывается устройство для чтения, объявляется буфер для хеш-суммы. Если устройство не удалось открыть, функция завершается. Далее идет блок кода:

```

... ::lseek(file_fd, HASH_OFFSET, SEEK_SET);
    ssize_t bytes = read(file_fd, &hash_buff, HASH_SIZE);
    ::close(file_fd);
    if(bytes != HASH_SIZE)
        return ret;
    ret = QByteArray(hash_buff, HASH_SIZE);
    return ret; }

```

В данном блоке устанавливается позиция с которой необходимо считывать данные в памяти устройства, считывается хеш-суммы и ее длина. Если длина хеш-суммы неверная, функция возвращает код работы и завершается. Если длина хеш-суммы верная, то она записывается в переменную и функция возвращает это значение.

2.3.6 Функция получения метки носителя.

Сначала в функции getLabel выполняются следующие команды:

```

int disktest::getLabel(TStDevInfo &devInfo, bool partition)
{
    int fd;
    int bufsize=BUFSIZE;
    unsigned char buff[bufsize];
    memset(buff, 0, sizeof(buff));
    if( -1 != (fd = ::open( devInfo.fileDev.data(), O_RDONLY)) )
    {

```

```
devInfo.devSize = ::lseek64(fd, 0, SEEK_END);
::lseek( fd, 0, SEEK_SET ); ...
```

В приведенном блоке инициализируются различные переменные, после чего производится попытка открыть устройство для чтения данных с него. Если открыть устройство удалось, в переменную записывается объем накопителя. Далее следует блок кода:

```
...      int n = ::read( fd, buff,  bufsize);
          ::close(fd);
          if(n==-1 && -1 != (fd = ::open( devInfo.fileDev.data(),
O_RDONLY | O_NONBLOCK)) )
          {
              n = ::read( fd, buff,  bufsize);
              ::close(fd);
          }
          if( bufsize != n )
          {
              syslog(LOG_WARNING, "Ошибка, %s:%d, чтение %s: %m",
__FILE__, __LINE__, devInfo.fileDev.data());
              QMessageBox::warning( 0, "Носитель", "Ошибка чтения
накопителя");
              return -1;
          }
      }
      else
      {
          syslog(LOG_WARNING, "Ошибка, %s:%d, открытие %s: %m",
__FILE__, __LINE__, devInfo.fileDev.data() );
          return -1;
      }
  }...
```

В приведенном блоке происходит закрытие ранее открытого устройства. Затем устройство открывается еще раз, но с использованием другого режима. Происходит чтение `mbr` (главной загрузочной записи). Если операция не удалась, выдается сообщение об ошибке чтения накопителя. Далее следует

исключение для условия, которое представлено в предыдущем блоке кода. В случае возникновения этого исключения выдается сообщение об ошибке открытия накопителя. Далее идет блок кода:

```
... struct ext2_super_block* sbb = (struct ext2_super_block*)
(buff+1024);
    if(devInfo.dev_type != VNI_CDROM)
    {
        if( 0 == memcmp("FAT12", buff+54, 5) )
            devInfo.fs_type = FAT12;
        else if( 0 == memcmp("FAT16", buff+54, 5) )
            devInfo.fs_type = FAT16;
        else if( 0 == memcmp("FAT32", buff+82, 5) )
            devInfo.fs_type = FAT32; ...
```

В данном блоке происходит инициализация структуры файловой системы ext2 для последующего использования. Затем, если текущее устройство не является диском, запускается определение файловой системы на устройстве. В данном блоке идет определение является ли файловая система на устройстве разновидностью файловой системы FAT. Далее идет блок:

```
... else if( sbb->s_magic[0] + 256*sbb->s_magic[1] ==
EXT2_SUPER_MAGIC )
    {
        if(sbb->s_feature_compat &
EXT3_FEATURE_COMPAT_HAS_JOURNAL)
            devInfo.fs_type = EXT3;
        else
            devInfo.fs_type = EXT2;
    }
    else if( buff[0] == 0xEB &&
            buff[1] == 0x52 &&
            buff[2] == 0x90 &&
            buff[3] == 0x4E && // N
            buff[4] == 0x54 && // T
            buff[5] == 0x46 && // F
```

```

        buff[6] == 0x53 )    // S
    devInfo.fs_type = NTFS;
}...

```

В данном блоке происходит дальнейшее определение файловой системы на устройстве. Происходит сравнение файловой системы на устройстве с файловыми системами ext2, ext3, ext4 и NTFS. Таким образом, по завершению данного блока файловая система на устройстве будет определена. Далее идет блок:

```

...if(!partition)
{
    syslog(LOG_WARNING, "Неверная файловая система на
накопителе %s", devInfo.fileDev.data());
    return -1;
}
switch( devInfo.fs_type )
{
case FAT12:
case FAT16:
    devInfo.label = QByteArray((const char*)buff+43, 11);
    break;
case FAT32:
    devInfo.label = QByteArray((const char*)buff+71, 11);
    break;
case EXT2:
case EXT3:
case EXT4:
    devInfo.label = QByteArray(sbb->s_volume_name, 16);
    break;
case NTFS:
    struct ntfs_param *
ntfs_prm=getNTFSParams(devInfo.fileDev.data());
    if(ntfs_prm!=NULL)
        devInfo.label = ntfs_prm->label.toAscii();
}
}

```

```

        break;
    }
    devInfo.reg = (devInfo.label[0]!='\0') && (devInfo.label[1]!
    ='\0');
    if(devInfo.reg)
        devInfo.label = devInfo.label.mid(1);

    if(devInfo.fs_type!=NTFS)
        devInfo.label = QTextCodec::codecForName("KOI8-R") -
>toUnicode(devInfo.label).toAscii();
    return 0;
}

```

Сначала в данном блоке идет исключение, которое сработает в случае, если файловая система на накопителе не поддерживается. Затем, в зависимости от файловой системы на устройстве, считывается определенный участок памяти накопителя в котором содержится метка. В зависимости от файловой системы изменяется кодировка полученных символов.

2.3.7 Функция заполнения списка устройств в графическом интерфейсе.

Сначала в функции fillTree выполняются следующие команды:

```

void disktest::fillTree(QTreeWidgetItem *root, QList<TStDevInfo>
partList, StorageClass storageClass)
{
    QTWIDev* pitem = 0;
    QTWIDev* item = 0;
    QByteArray pass_dev;
    for(int i = 0; i < partList.count(); i++)
    {
        if(!pass_dev.isEmpty() &&
partList.at(i).fileDev.startsWith(pass_dev))
            continue;
        if(partList.at(i).fileDev.right(1).toInt() != 0 && pitem)
        {
            item = new QTWIDev(pitem);

```

```

        item->setText(0, "Раздел");
        pitem->setExpanded(true);
        item->setIcon(0, QIcon(":/images/kcmpartitions.png"));
    }
else
{
    item = new QTWIDev(root);
    pitem = item;
    item->setText(0, partList.at(i).model);
    if(partList.at(i).removable)
        item->setIcon(0, QIcon(":/images/media-usb-
flash.png"));
    else
        item->setIcon(0, QIcon(":/images/drive-
harddisk.png"));
}...

```

В приведенном блоке запускается цикл перебора всех элементов списка разделов. Если элемент на текущей итерации распознается как раздел какого-либо носителя, то ему присваивается соответствующая иконка, которая в дальнейшем будет отражена в таблице устройств графического интерфейса. Далее, если текущий элемент не является разделом (если на накопителе нет деления памяти на разделы), происходит считывание модели устройства. Затем, если устройство является съемным носителем, то элементу присваивается изображение flash-устройства в графическом интерфейсе программы, иначе присваивается изображение жесткого диска. Далее идет блок кода:

```

...if(storageClass == StorageClassLocal)
    item->setText(1,
QFileInfo(partList.at(i).fileDev).fileName());
    if(!partList.at(i).serialNumber.isEmpty())
    {
        int res = checkSerialHash(partList[i]);
        switch(res)
        {

```

```

        case 0:
            syslog(LOG_INFO, "%s", QString("Проверка
устройства %1 пройдена
успешно!").arg(partList.at(i).fileDev.data()).toAscii().data());
            item->setIcon(2, QIcon(":/images/vni_check.png"));
            break;
        case 1:
            syslog(LOG_INFO, "%s", QString("Ошибка! Неверный
криптографический ключ на устройстве
%1!").arg(partList.at(i).fileDev.data()).toAscii().data());
            item->setIcon(2, QIcon(":/images/vni_warn.png"));
            break;
        case 2:
            item->setIcon(2, QIcon(":/images/vni_warn.png"));
            break;
            item->setIcon(2, QIcon(":/images/vni_warn.png"));
        default:
            break;
    }
}
else if(partList[i].removable)
    item->setIcon(2, QIcon(":/images/vni_warn.png"));...

```

В данном блоке кода проверяется есть ли серийный номер у данного устройства. Если серийный номер есть, то происходит вызов функции проверки хеш-суммы на накопителе `checkSerialHash`, которая будет рассмотрена далее в работе. В зависимости от результата работы вызванной функции, будет выдано сообщение об успехе операции или об ошибке. Также, в зависимости от результата проверки хеш-суммы, в графическом интерфейсе носителю будет присвоено то или иное изображение, отражающее результат проверки. Если серийного номера у устройства нет, то в графическом интерфейсе носителю присваивается изображение, сигнализирующее об ошибке. Далее идет блок кода:

```

...      item->setText(2, partList.at(i).serialNumber);
          item->setText(3,
QVariant((qulonglong)partList.at(i).devSize/1024/1024).toString());
          item->setTextAlignment(3, Qt::AlignRight);
          item->setText(4, fs_types[partList.at(i).fs_type]);
          item->setTextAlignment(4, Qt::AlignCenter);
          item->setText(5, partList.at(i).label);
          item->setData(0, Qt::UserRole, storageClass);
          item->m_devInfo = partList.at(i);
      }
      ui->treeWidget->resizeColumnToContents(0);
      ui->treeWidget->resizeColumnToContents(1);
      ui->treeWidget->resizeColumnToContents(2);
      ui->treeWidget->resizeColumnToContents(3);
      ui->treeWidget->resizeColumnToContents(4);
      ui->treeWidget->resizeColumnToContents(5);
      if(ui->treeWidget->topLevelItemCount() > 0)
      {
          ui->treeWidget->itemAt(0, 0)->setSelected(true);
          ui->treeWidget->setCurrentItem(ui->treeWidget->itemAt(0,
0));
      }
  }
}

```

В данном блоке заполняются оставшиеся столбцы таблицы графического интерфейса ранее полученными данными об устройстве. Затем происходит автонастройка ширины столбцов таблицы по содержащимся в них данным.

2.3.8 Функция проверки хеш-суммы.

Сначала в функции checkSerialHash выполняются следующие команды:

```

int disktest::checkSerialHash(TStDevInfo devInfo)
{
    if (!devInfo.serialNumber.isEmpty())
    {
        QByteArray hash = devInfo.hash;
    }
}

```

```

        QByteArray serialNumber = devInfo.serialNumber;
        char* hash_from_serial = passwordhash(serialNumber.data(),
hash.data());...

```

В приведенном блоке определяется есть ли серийный номер у устройства, если серийного номера у устройства нет, то функция завершает свою работу, возвращая единицу. Если серийный номер есть, то в переменные записываются хеш-сумма и серийный номер с устройства. Затем происходит проверка хеш-суммы с помощью функции passwordhash, которая будет описана далее в работе. Далее идет блок кода:

```

...         if(hash_from_serial) {
                if(strncmp(hash_from_serial, hash.data(), HASH_SIZE)){
                        syslog(LOG_INFO, "%s", QString("Ошибка! Неверный
криптографический ключ На устройстве
%1!").arg(devInfo.fileDev.data()).toAscii().data());
                        return 1; }
                else {
                        syslog(LOG_INFO, "%s", QString("Проверка
устройства %1 пройдена
успешно!").arg(devInfo.fileDev.data()).toAscii().data());
                        return 0;
                }
        } else {
                //syslog(LOG_INFO, "%s", QString("Не удалось выполнить
проверку криптографического ключа на устройстве
%1").arg(devInfo.fileDev.data()).toAscii().data());
                return 2;
        } }
return 1; }

```

В приведенном блоке проверяется результат работы функции passwordhash. Если функция закончила свою работу без ошибки, то полученный результат сравнивается с хеш-суммой на устройстве. Если хеш-суммы сходятся, то в журнал отладки записывается сообщение об успешности проверки

устройства и функция возвращает «0». Если хеш-суммы не сходятся, то в журнал записывается сообщение об ошибке и функция возвращает «1». Если же функция passwordhash завершилась с ошибкой, то текущая функция возвращает «2».

2.3.9 Функция вычисления хеш-суммы.

В функции passwordhash выполняются следующие команды:

```
char* passwordhash(char* text, char* keyset)
{
    unsigned int ret;
    static char result[13];
    char* salt = keyset;
    char* kzuinit = (char*)kzu;
    if(!strlen(text) || !strlen(keyset))
        return NULL;
    if(keyset[0] == '$')
        salt++;
    kzuinit[0] = salt[0];
    kzuinit[1] = salt[1];
    ret = giz((unsigned char*)text, strlen(text),0); //контрольная
сумма
    sprintf(result, "$%.2s$%X\0", salt, ret); // $, преобразование
2ух символов из салт, $, 16рич знач retS

    return result;
}
```

В приведенном блоке сначала инициализируются переменные, которые будут необходимы в процессе работы данной функции. Затем проверяется, являются ли полученные на входе значения переменной-буфера данных и ключа пустыми, если да, то функция завершается и возвращает значение NULL. Если в полученном ключе нулевой символ это символ «\$», то это значит, что функции на вход поступила хеш-сумма для расшифровки. В этом случае чтение переменной-ключа начинается не с нулевого, а с первого символа. Если нулевой

символ не «\$», то это значит, что на вход поступила соль и необходимо зашифровать полученные данные. Далее вызывается функция `giz_3`, реализующая ГОСТ 28147-89 в режиме выработки имитовставки [13]. После получения результата работы функции `giz_3` в переменную результата записывается значение формата: символ «\$», 2 символа выработанной ранее соли, символ «\$», результат работы функции `giz_3` в шестнадцатеричной системе счисления. Данная переменная является результатом работы текущей функции.

2.3.10 Функция, реализующая ГОСТ 28147-89 в режиме выработки имитовставки.

Сначала в функции `giz_3` выполняются следующие команды:

```
unsigned int giz_3( unsigned char* text, int size, unsigned char*
mask )
{
    unsigned int N1, N2;
    unsigned int CM1, CM2, R;

    int i, k;
    unsigned int *otl;
    unsigned int *m;
    unsigned int tail[2] = {0,}; // массив из 2 элементов (второй
пустой)
    unsigned int tail_mask[2] = {0,};
    otl = (unsigned int*)text; // записывается буфер данных
    m = (unsigned int*)mask; // записывается маска накрывающая
данные
    if( size%8 ) //если размер данных текста не кратен 8
    {
        memcpy( (char*)tail, text+((size/8)*8), size -
( (size/8)*8) );
        if(mask)
            memcpy( (char*)tail_mask, mask, size -
```

```
( (size/8)*8) );
    }...
```

В приведенном блоке объявляются переменные для дальнейшего использования в функции. Затем записывается буфер открытых данных, записывается маска. Затем проверяется, кратны ли данные восьми. В случае, если остаток от деления на 8 больше нуля, эти данные записываются в отдельную переменную. Далее идет блок:

```
...    k = 0;
        N1 = 0;
        N2 = 0;
        while(size > 0)
        {
            if( size > 7 )
            {
                N1 = mask?N1^ot1[k]^m[0]:N1^ot1[k];
                N2 = mask?N2^ot1[k+1]^m[1]:N2^ot1[k+1];
            }
            else
            {
                N1 = mask?N1^tail[0]^tail_mask[0]:N1^tail[0];
                N2 = mask?N2^tail[1]^tail_mask[1]:N2^tail[1];
            }
        }...
```

В данном блоке обрабатывается массив основного текста и переменная, в которую был записан остаток от деления на 8 ранее. В случае, если на данные наложена маска, она снимается. Далее идет блок кода:

```
...    int i_16, j_16;
        for(i_16 = 0; i_16 < 2; i_16++)
        {
            for(j_16 = 0; j_16 < 8; j_16++)
            {
                { // zik11(j);
                    int m,n,l;
```

```
CM1 = N1 + kzu[j_16];
R = 0L;...
```

В данном блоке открывается два цикла функции, которые представляют собой 32 цикла шифрования ГОСТ 28147-89 в режиме простой замены [13]. Происходит сложение содержимого переменной N1 с данными из контрольного запоминающего устройства по модулю 2 в переменной-сумматоре CM1. Далее идет блок кода:

```
...          for(m = 7; m >= 0; m--)
              {
                  n = 4*m;
                  l = (CM1>>n) & 0xf;
                  R = R<<4;
                  R = R^zamena[l][7-m];
              }...
```

В данном блоке происходит разбиение результата на восемь четырех битовых последовательностей, каждая из которых поступает на вход своего угла таблицы замен. Происходит сдвиг бит переменной на n разрядов, в которой находится сложение по модулю переменной N и два данных и значений контрольного запоминающего устройства. После этого происходит обнуление всех бит, кроме последних четырех, которые сохраняются в отдельную переменную. Затем происходит сдвиг другой переменной на четыре бита влево, таким образом последние четыре бита переменной будут заполнены нулями. Далее эта переменная записывается результатом сложения по модулю значения текущей переменной и значением из соответствующей строки и столбца таблицы узла замены. Далее идет блок кода:

```
...          m = R>>21;
              R = (R<<11) + m;
              CM2 = R^N2;
              } // zikl1(j);

          N2 = N1;
```

```

        N1 = CM2;
    }
}
} // zikl16();
k += 2;
size -= 8;
}
return (N1);}

```

В приведенном блоке происходит сдвиг бит в ранее полученной переменной на 21 разряд вправо и запись в переменную *m*. После этого, ранее полученная переменная регистра сдвига *R* перезаписывается значением переменной *R* со сдвигом на 11 бит влево и сложением с переменной *m*. Затем происходит запись в промежуточную переменную-сумматор *CM2* результат побитного сложения по модулю 2 переменных регистра сдвига *R* и накопителя *N2*. После завершения одного из циклов заполнение накопителя *N1* переписывается в накопитель *N2*, результат с выхода сумматора *CM2* переписывается в накопитель *N1*. После этого первый цикл заканчивается. После тридцать второго цикла в *N1* информация сохраняется, а результат с выхода сумматора *CM2* переписывается в *N2*. Данные в *N1* будут являться блоком зашифрованных данных.

2.3.11 Функция запуска форматирования накопителя.

Сначала в функции `formatting_local_drive` выполняются следующие команды:

```

void disktest::formatting_local_drive()
{
    const char* command = CHROOT"/fdrive/fdrive";
    QTWIDev* twi = (QTWIDev*)ui->treeWidget->currentItem();
    if(access(command, X_OK))
    {
        QMessageBox::critical(this, "Ошибка", QString("not found
%1").arg(command));
    }
}

```

```

        this->update();
        return;
    }
    if(!twi)
        return ;
    QTWIDev* root_parent = twi;
    while(root_parent->parent())
        root_parent = (QTWIDev*)root_parent->parent();
    char *hash_buff = NULL;
    if(twi->parent())
        hash_buff = hashSerialNumber(((QTWIDev *)root_parent)-
>m_devInfo.serialNumber.data());
    else
        hash_buff = hashSerialNumber(twi-
>m_devInfo.serialNumber.data());...

```

В приведенном блоке путь к проекту форматирования fdrive записывается в переменную, если проект по данному пути не найден, выдается ошибка. Затем происходит вызов функции шифрования hashSerialNumber, результат работы функции записывается в переменную-буфер. Далее идет блок кода:

```

...    QProcess process;
        process.start(command, QStringList() << twi->m_devInfo.fileDev
<< "1");
        if(!process.waitForStarted()) {
            const char *msg = "Не удалось запустить
приложение \"Форматирование накопителя\"";
            syslog(LOG_INFO, msg);
            QMessageBox::critical(this, "Ошибка", msg);

            return ;
        }
        process.waitForFinished(-1);

        if((process.exitStatus() != QProcess::NormalExit) ||

```

```

(process.exitCode()))
{
    syslog(LOG_ERR, "Ошибка, %s:%d, Некорректное завершение
приложения %s: %m", __FILE__, __LINE__, "Форматирование
накопителя");
    QMessageBox::critical(this, "Ошибка", "Некорректное
завершение приложения " + QString("Форматирование накопителя"));
    return ;
}...

```

В приведенном блоке кода вызывается проект fdrive на выполнение. Если запустить проект не удалось, выдается соответствующая ошибка и текущая функция завершается. Если проект запустить удалось, ожидаем завершения его выполнения. Если проект был запущен, но был завершен из-за ошибки, выдается соответствующее сообщение. Далее идет блок кода:

```

...    int repart_fd = ::open(root_parent->m_devInfo.fileDev.data(),
O_RDWR);
    if(repart_fd != -1)
    {
        write_hash(repart_fd, hash_buff);

        if(ioctl(repart_fd, BLKRRPART, 0) == -1)
        {
            syslog(LOG_ERR, "Ошибка, %s:%d, ioctl(repart_fd,
BLKRRPART, 0) %s: %m", __FILE__, __LINE__, twi-
>m_devInfo.fileDev.data());
            QMessageBox::critical(this, "Ошибка", "Ошибка
обновления таблицы разделов");
        }
        ::close(repart_fd);
    }
}

```

В данном блоке кода открывается текущее устройство для дальнейшей работы с ним. Если устройство удалось открыть, вызывается функция записи

хеш-суммы на накопитель `write_hash`, которая будет описана далее в работе. После выполнения данной функции проверяется, что таблицы разделов на устройстве были обновлены, выдается сообщение об ошибке, если этого не произошло.

2.3.12 Функция шифрования.

В функции `hashSerialNumber` выполняются следующие команды:

```
char* disktest::hashSerialNumber(char *serialNumber)
{
    char thesalt[3];
    unsigned char mas[2];
    VYR_GSCH(mas, 2);
    thesalt[0] = mksalt(mas[0]);
    thesalt[1] = mksalt(mas[1]);
    thesalt[2] = '\\0';
    char* crypt_password = passwordhash(serialNumber, thesalt);
    return crypt_password;
}
```

В приведенном блоке сначала объявляются массивы для дальнейшего использования в функции. Затем вызывается функция генерации случайных чисел `VYR_GSCH`, которая будет описана далее в работе. После этого происходит заполнение переменной-массива результатами работы функции получения символа из числовой последовательности `mksalt`, которая будет описана далее в работе. Затем происходит вызов функции вычисления хеш-суммы `passwordhash`, которая была описана ранее в работе, полученный результат возвращается текущей функцией.

2.3.13 Функция генерации случайных чисел

В функции `VYR_GSCH` выполняются следующие команды:

```
int disktest::VYR_GSCH(unsigned char* buf, unsigned int count)
{
    int i;
    srand(time(NULL));
```

```

    for(i = 0; i<count; i++)
        buf[i] = rand();
    return 0;
}

```

В приведенном блоке происходит выработка необходимого количества случайных чисел, которые записываются в переменную-массив.

2.3.14 Функция получения символа из числовой последовательности.

В функции `mksalt` выполняются следующие команды:

```

char disktest::mksalt(int seed)
{
    int num = seed % 64;
    if (num < 26)
        return 'a' + num;
    else if (num < 52)
        return 'A' + (num - 26);
    else if (num < 62)
        return '0' + (num - 52);
    else if (num == 63)
        return '.';
    else
        return '/';
}

```

В приведенном блоке сначала происходит получение остатка от деления на 64 полученного на входе значения. Затем остаток сравнивается с различными величинами, и, в зависимости от значения остатка от деления, происходит вывод символа из того или иного набора символов.

2.3.15 Функция записи хеш-суммы на накопитель.

В функции `write_hash` выполняются следующие команды:

```

void disktest::write_hash(int fd, const char *hash)
{
    if(!hash)
        return ;
}

```

```

::lseek(fd, HASH_OFFSET, SEEK_SET);
ssize_t wr = ::write(fd, hash, HASH_SIZE);
if(wr < 0) {
    int err = errno;
    syslog(LOG_INFO, "Ошибка записи криптографической метки на
носитель: %m (%d)", err);
}
}

```

В приведенном блоке происходит проверка была ли получена на входе в функцию хеш-сумма. Если нет, то функция завершает свою работу, если да, то работа функции продолжается. Затем на устройстве выделяется участок памяти, в которой записывается полученная хеш-сумма. Если во время записи произошла ошибка, то выдается соответствующее сообщение.

2.4 Разработка и описание алгоритма программы, отвечающей за форматирование

Был разработан алгоритм проекта «fdrive», который отвечает за форматирование устройства и создание с помощью встроенной утилиты Linux «msdosfs» файловой системы на накопителе. Проект включает в себя следующие основные функции:

- функция инициализации основного окна приложения «init»;
- функция обработки нажатия на кнопку форматирования «on_b_format_clicked»;
- функция обработки завершения процесса «procFinished».

При вызове данного проекта при работе проекта disktest, который был описан ранее, появляется окно графического интерфейса, в котором находятся следующие элементы:

- поле для ввода метки носителя;
- опция быстрого форматирования;

- кнопка «Форматировать»;
- кнопка «Заккрыть».
- шкала прогресса выполнения форматирования.

При появлении графического интерфейса поле для метки будет заполнено по умолчанию текстом «Flash». Опция быстрого форматирования будет по умолчанию включена, если ее отключить, то форматирование накопителя произойдет с проверкой устройства на поврежденные блоки памяти, что займет значительно больше времени, чем быстрое форматирование. Кнопка «Форматировать» запустит работу проекта. Нажатие на кнопку «Заккрыть» завершит работу проекта. На шкале после нажатия кнопки «Форматировать» будет отображаться прогресс выполнения форматирования в процентах.

На рисунке 18 представлен алгоритм функции инициализации основного окна приложения «init».

На рисунке 19 представлен алгоритм функции обработки нажатия на кнопку форматирования «on_b_format_clicked».

На рисунке 20 представлен алгоритм функции обработки завершения процесса «procFinished».

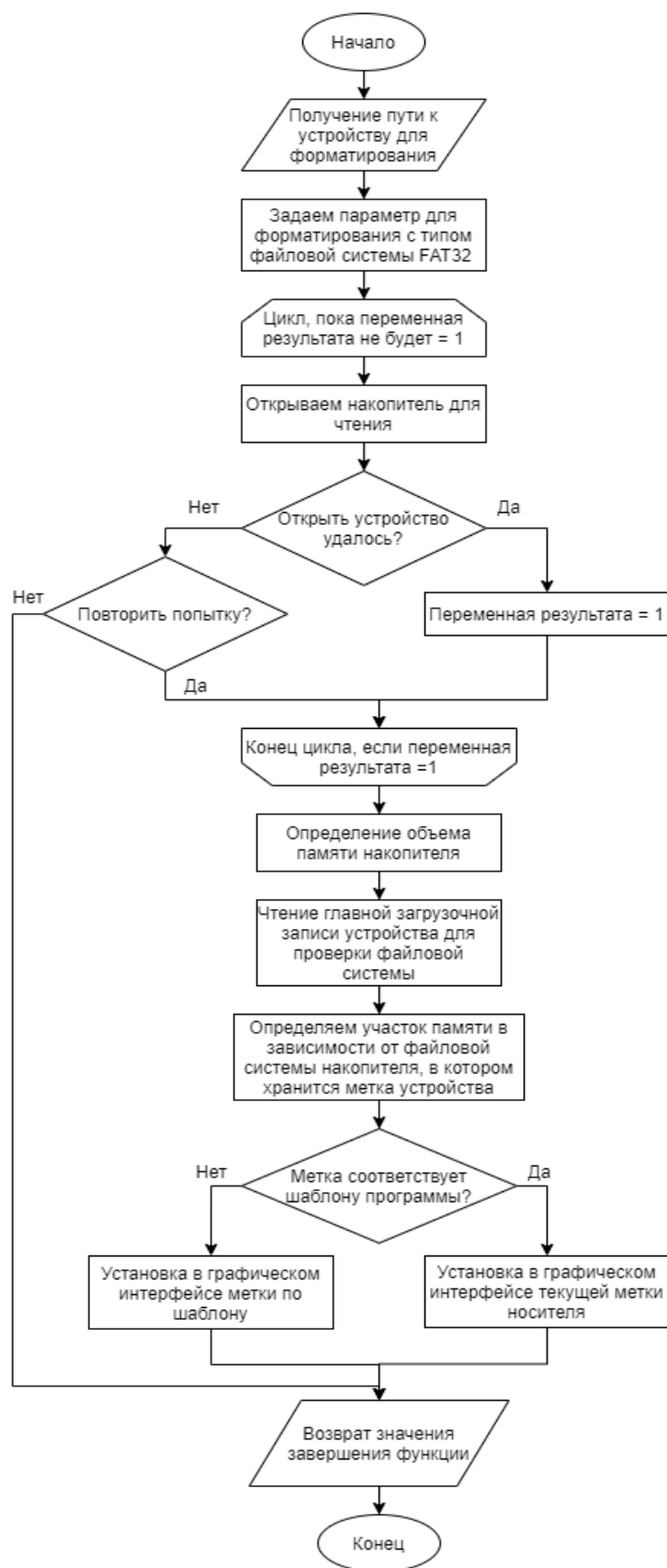


Рисунок 18 — Алгоритм функции инициализации основного окна приложения «init»

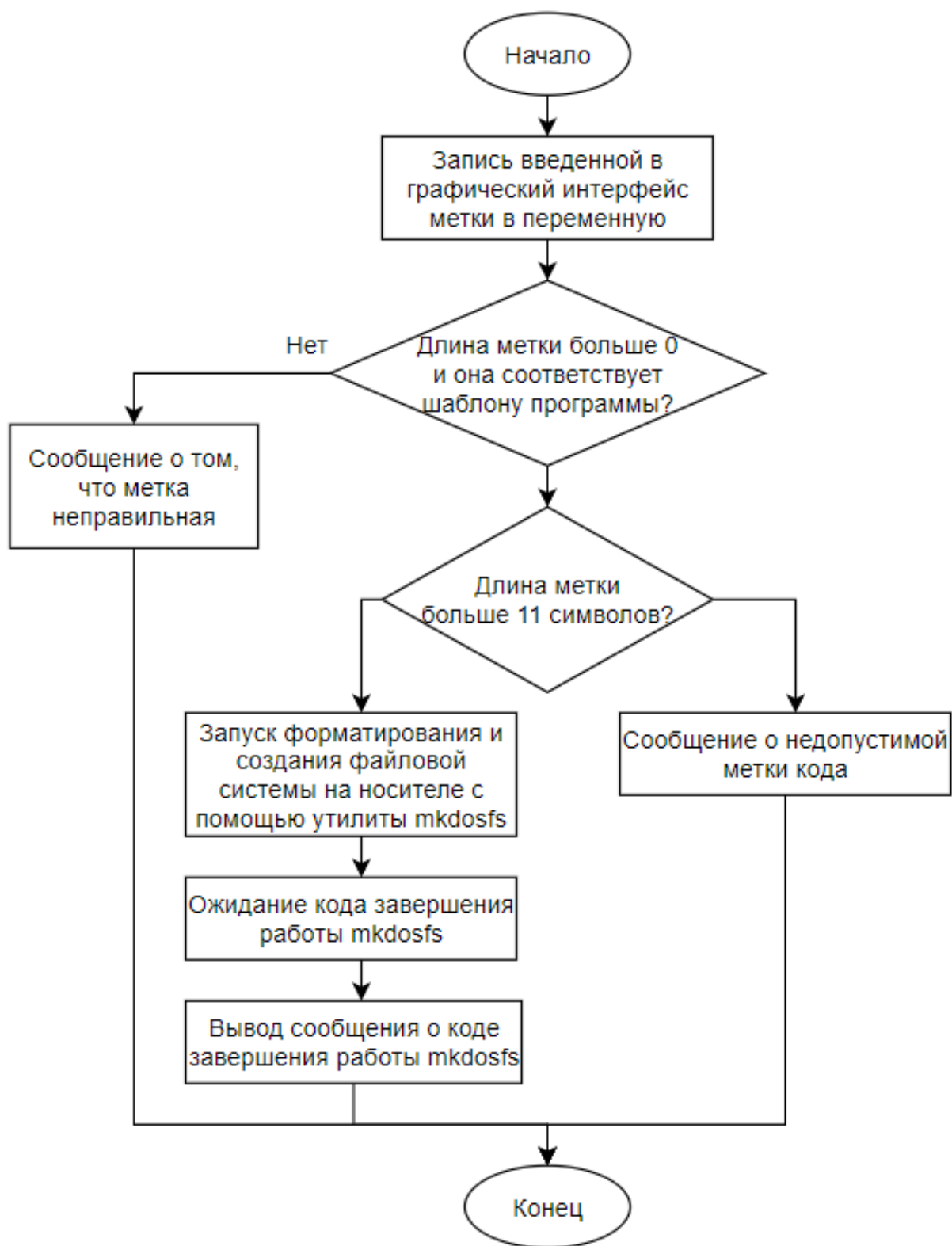


Рисунок 19 — Алгоритм функции обработки нажатия на кнопку форматирования «on_b_format_clicked»

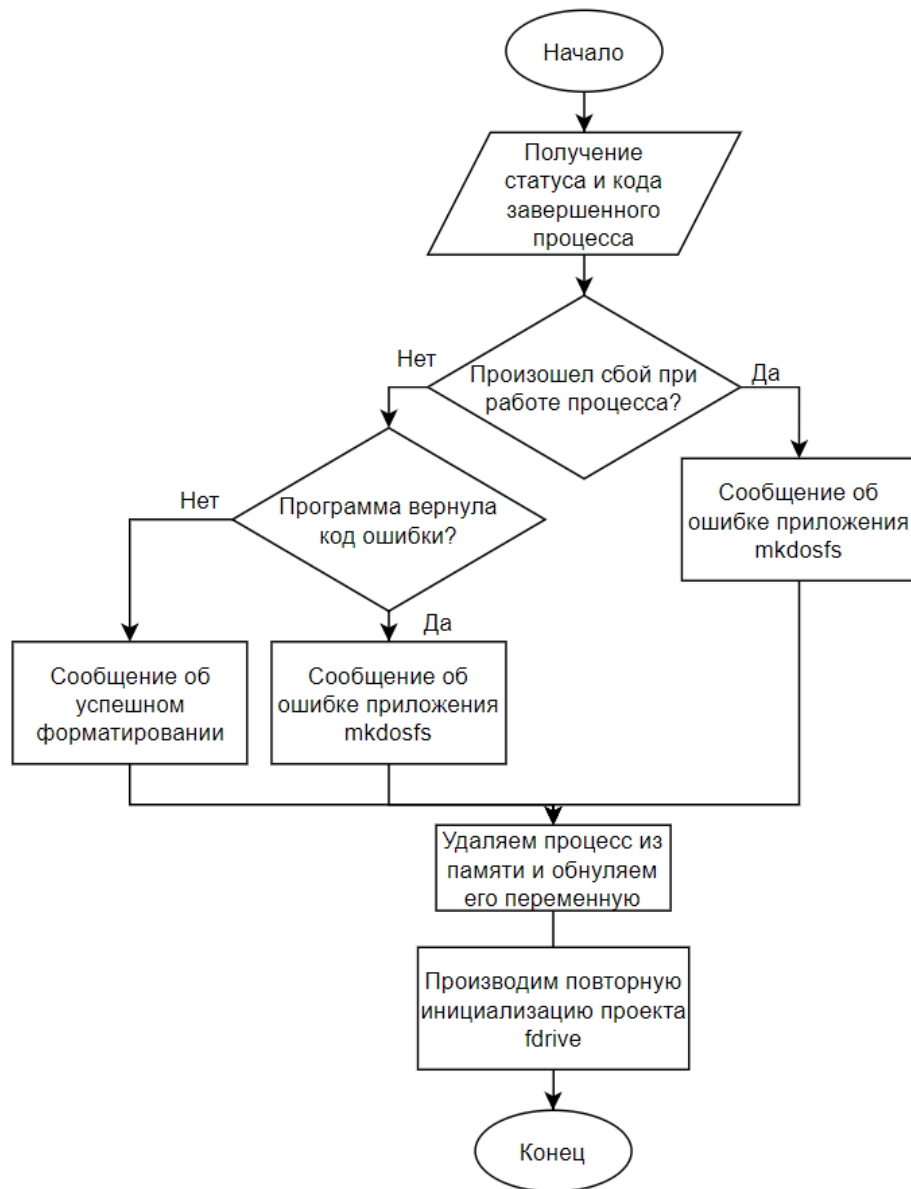


Рисунок 20 — Алгоритм функции обработки завершения процесса
«procFinished»

2.5 Разработка программы, отвечающей за форматирование устройства

2.5.1 Описание начальных переменных

Был рассмотрен заголовочный файл fdrive.h:

```

namespace Ui {
    class FDrive;
}

class FDrive : public QDialog {

```

```

    Q_OBJECT
public:
    FDrive(QWidget *parent = 0);
    ~FDrive();

    int init(QString dev, bool vnio = false);
    void fEnabled(bool);

protected:
    void changeEvent(QEvent *e);

private:
    QString _dev;
    QString m_format;
    QStringList m_arg;
    QProcess *m_proc;
    int UEvent;
    int max_size;
    bool m_vnio;

    Ui::FDrive *ui;
    enum {UNKNOWN = 0, FAT12, FAT16, FAT32};

private slots:
    void on_b_close_clicked();
    void on_b_format_clicked();

    void procFinished(int, QProcess::ExitStatus);
    void procError(QProcess::ProcessError);
    void procRRStdErr();
    void procRRStdOut();

protected slots:

```

```
void closeEvent(QCloseEvent*);
};
```

Сначала объявляется класс, который используется в данном проекте, затем идет описание переменных программы и соответствующие им типы, а также функции и типы этих функций, которые также используются в данном проекте. Далее приводится перечисление используемых файловых систем в программе, затем идет перечисление функций-слотов, которые обрабатывают различные события при работе проекта, например нажатие на кнопку графического интерфейса.

2.5.2 Функция инициализации основного окна приложения.

Сначала в функции `init` выполняются следующие команды:

```
int FDrive::init(QString dev, bool vnio)
{
    QByteArray m_label;
    _dev = dev;
    m_arg.clear();
    m_arg << "-I" << "-F" << "32";

    int fd, nResult, fs = UNKNOWN;
    char buff[BUFFSIZE];

    ui->progressBar->setMinimum(0);

    do
    {
        fd = ::open( dev.toAscii(), O_RDONLY );
        nResult = 1;
        if( -1 == fd )
            nResult = QMessageBox::warning ( this, aName, "Не могу
открыть накопитель для чтения", "&Повторить", "Отмена" );
    }
    while( !nResult );
```

```

if( -1 == fd )
{
    ::close(fd);
    return 0;
}...

```

В приведенном блоке функции на вход поступает путь к устройству, с которым в дальнейшем необходимо будет проводить работу, путь записывается в переменную. Затем задается ключ для утилиты Linux `mkdosfs`, который подразумевает, что форматирование будет производиться с созданием файловой системы на устройстве FAT32. Далее открываем устройство для чтения, переводя его данные в необходимую для функции кодировку. Если устройство открыть не удалось, выдается соответствующее сообщение, в котором возможно нажать кнопку повтора попытки чтения устройства «Повторить». Цикл будет выполняться, пока устройство не будет определено, либо пока пользователь не нажмет кнопку «Отмена» в появившемся окне предупреждения о невозможности чтения устройства. В этом случае функция завершит свою работу. Далее идет блок кода:

```

...    off64_t res_size = ::lseek64(fd, 0, SEEK_END);
    max_size = res_size >> 10;
    qWarning ("max_size - %d", max_size);
    ui->progressBar -> setMaximum(max_size);

    ::lseek( fd, 0, SEEK_SET );
    //Читаем mbr для проверки файловой системы
    if( BUFSIZE != ( nResult = ::read( fd, buff, BUFSIZE) ) )
    {
        QMessageBox::warning( this, aName, "Не могу прочитать
mbr", QMessageBox::Ok, QMessageBox::NoButton );
        ::close(fd);
        return 0;
    }...

```

В данном блоке выполняется определение объема памяти устройства, далее читаем участок памяти с главной заголовочной записью, в которой содержатся данные о текущей файловой системе устройства. Далее идет блок кода:

```
...    if( 'F' != buff[54] || 'A' != buff[55] || 'T' != buff[56] ||
'1' != buff[57] || '2' != buff[58] )
    {
        if( 'F' != buff[54] || 'A' != buff[55] || 'T' != buff[56]
|| '1' != buff[57] || '6' != buff[58] )
        {
            if( 'F' != buff[82] || 'A' != buff[83] || 'T' !=
buff[84] || '3' != buff[85] || '2' != buff[86] )
                fs = UNKNOWN;
            else
                fs = FAT32;
        }
        else
            fs = FAT16;
    }
else
    fs= FAT12;
if( FAT32 == fs )
{
    buff[82] = '\0';
    m_label = QByteArray(buff + 71, 11);
}
else
{
    buff[54] = '\0';
    m_label = QByteArray(buff + 43, 11);
}...
```

В приведенном блоке идет определение файловой системы на устройстве путем сравнения символов из главной загрузочной записи с символами из

названия файловых систем. Далее, после определения файловой системы, происходит считывание метки носителя в переменную. Далее идет блок кода:

```
... if(vnio)
{
    QRegExp regexp("Flash\\d{0,}");
    QRegExpValidator *validator = new QRegExpValidator(regexp,
ui->lineEdit);
    m_vnio = true;

    ui->lineEdit->setValidator(validator);
    int pos = 0;
    QString label = QString(m_label);

    if(validator->validate(label, pos) ==
QValidator::Acceptable)
        ui->lineEdit->setText(m_label);
    else
        ui->lineEdit->setText("Flash");
}
else
{
    if(!m_label.isEmpty())
        ui->lineEdit->setText(m_label);
    else
        ui->lineEdit->setText("NO NAME");
}

return 1;
}
```

В приведенном блоке полученная метка носителя сравнивается с шаблоном метки, заданным в программе. Если метка с шаблоном совпадают, то в поле метки в графическом интерфейсе проекта записывается текущая метка с

устройства. Если метка не совпадает с шаблоном, то в графическом интерфейсе в поле метки вписывается шаблон.

2.5.3 Функция обработки нажатия на кнопку форматирования.

Сначала в функции `on_b_format_clicked` выполняются следующие команды:

```
void FDrive::on_b_format_clicked()
{
    if(m_vnio)
    {
        QRegExp regexp("Flash\\d{0,}");
        QRegExpValidator validator(regexp, 0);
        int pos = 0;
        QString label = ui->lineEdit->text();

        if ((label.length() <= 0) ||
            (validator.validate(label, pos) !=
QValidator::Acceptable))
        {
            QMessageBox::critical(this, "Ошибка", "Введена
неправильная метка");
            reject();
            return ;
        }
    }
    ...
}
```

В приведенном блоке текст из поля метки графического интерфейса записывается в переменную и затем сравнивается с шаблоном метки, заданным в программе. Если метка не соответствует шаблону, то выдается сообщение о том, что введена неправильная метки, и функция завершает свою работу. В ином случае выполнение функции продолжается. Далее идет блок кода:

```
...    if (ui->lineEdit->text().length() > 11)
    {
        QMessageBox::warning(this, aName,    // "Формат Диск " +
```

```
cmbBox1->currentText(),
```

```
"Указана недопустимая метка тома.
```

```
Введите правильную метку тома.", "&Заккрыть");
```

```
    fEnabled(TRUE);
```

```
    ui->lineEdit->selectAll();
```

```
    ui->lineEdit->setFocus();
```

```
    }...
```

В приведенном блоке происходит сравнение длины введенной метки. Если длина метки больше, чем одиннадцать символов, то выдается сообщение, что введена недопустимая метка. Далее идет блок кода:

```
... else
```

```
{
```

```
    if (!_dev.isEmpty() && !_dev.isNull())
```

```
    {
```

```
        m_proc = new QProcess(this);
```

```
        if (m_proc)
```

```
        {
```

```
            ui->progressBar->setValue(max_size / 100);//
```

первый шаг прогресс бара на 1 процент

```
            update();
```

```
            QApplication->processEvents();
```

```
            if(ui->checkBox->checkState() == Qt::Unchecked)
```

```
                m_arg << "-c";
```

```
            m_arg << QString("-n");
```

```
            m_arg << ui->lineEdit->text();
```

```
            m_arg << _dev;
```

```
            connect(m_proc,
```

```
SIGNAL(finished(int,QProcess::ExitStatus)), this,
```

```
SLOT(procFinished(int,QProcess::ExitStatus)));
```

```
            connect(m_proc,
```

```
SIGNAL(error(QProcess::ProcessError)), this,
```



```

void FDrive::procFinished(int exit_code, QProcess::ExitStatus
exit_status)
{
    disconnect(m_proc, SIGNAL(finished(int,QProcess::ExitStatus)),
this, SLOT(procFinished(int,QProcess::ExitStatus)));
    disconnect(m_proc, SIGNAL(error(QProcess::ProcessError)),
this, SLOT(procError(QProcess::ProcessError)));
    disconnect(m_proc, SIGNAL(readyReadStandardError()), this,
SLOT(procRRStdErr()));
    disconnect(m_proc, SIGNAL(readyReadStandardOutput()), this,
SLOT(procRRStdOut()));
    switch (exit_status)
    {
    case QProcess::CrashExit:
    {
        QMessageBox::critical(this,aName,QString("Ошибка запуска
приложения mkdosfs"));
        //syslog?
        fEnabled(TRUE);
        if(m_proc)
            delete m_proc;
        m_proc = 0;
        init(_dev);
        reject();
        break;
    }
    case QProcess::NormalExit:
    {
        if(exit_code != 0)
        {
            QMessageBox::critical(this,aName,QString("Ошибка
запуска приложения mkdosfs"));
            fEnabled(TRUE);
            if(m_proc)

```

```

        delete m_proc;
m_proc = 0;
init(_dev);
reject();
}
else{
    qWarning("Normal exit");
    fEnabled(TRUE);
    if(m_proc)
        delete m_proc;
    m_proc = 0;
    QMessageBox::information(this, aName, "Форматирование
завершено");
    ui->progressBar->setValue(0);
    init(_dev);
    accept();
}
break; }}}

```

В приведенном блоке обрабатываются статусы с которыми завершилась работа утилиты `mkdosf`. Существует два варианта исхода: процесс аварийно завершился, процесс завершился в стандартном режиме. В случае аварийного завершения и в случае, когда процесс завершился в стандартном режиме, но была возвращена ошибка, выдается сообщение о том, что произошла ошибка запуска приложения. В случае, если процесс завершился в стандартном режиме и процесс не возвратил сообщение об ошибке, выдается сообщение, что форматирование прошло успешно. Во всех перечисленных случаях, если процесс был создан, он удаляется и обнуляется. Затем происходит повторная инициализация работы проекта `fdrive` с помощью функции `init`, которая была описана ранее.

3 Экспериментальная проверка работоспособности программы

Для проверки работоспособности программы в качестве отчуждаемого носителя информации было использовано USB-flash устройство с объемом памяти 15268 мегабайт. На устройстве используется файловая система FAT32, перед использованием устройство было отформатировано стандартными средствами операционной системы, метка тома не использовалась.

Была запущенная программа и нажата кнопка «Обновить». После обновления таблицы устройств таблица осталась пуста. Результат представлен на рисунке 21.

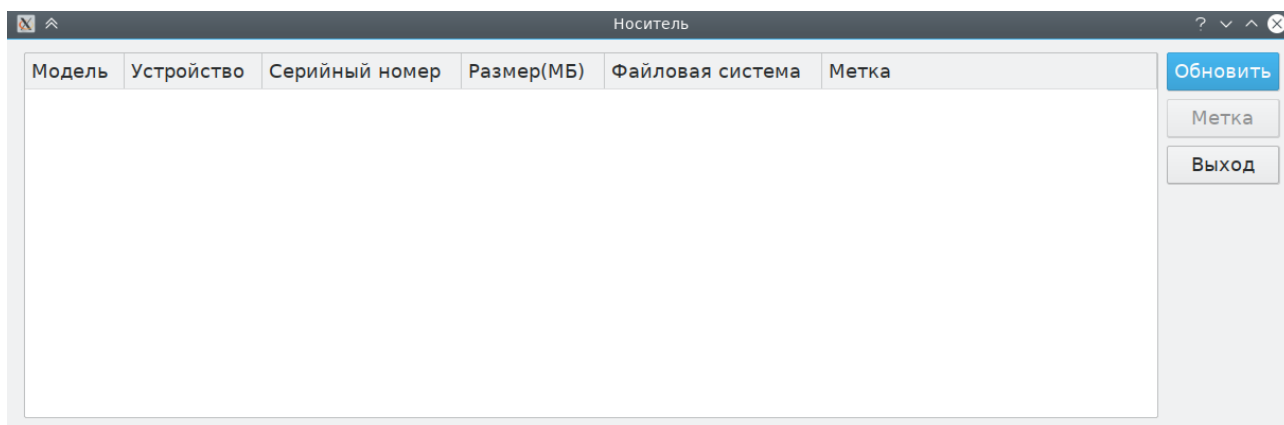


Рисунок 21 — Результат работы программы после обновления списка устройств без подключенного USB устройства

Затем отчуждаемый носитель был подключен к вычислительному устройству и вновь была нажата кнопка «Обновить». В результате устройство было определено и в таблице была заполнена одна строка с данным устройством. В столбце «Серийный номер» появилось изображение, которое говорит о том, что на устройстве не была найдена верная хэш-сумма. Результат представлен на рисунке 22. Далее была нажата кнопка «Метка», в появившихся окнах предупреждения были нажаты кнопки «ОК» и «Продолжить», затем в окне запуска процедуры форматирования была нажата кнопка «Форматировать» с выбранным режимом «Быстрое форматирование». Форматирование заняло

около секунды, после его завершения было выдано сообщение об успешном завершении процедуры. Результат представлен на рисунках с 23 по 26.

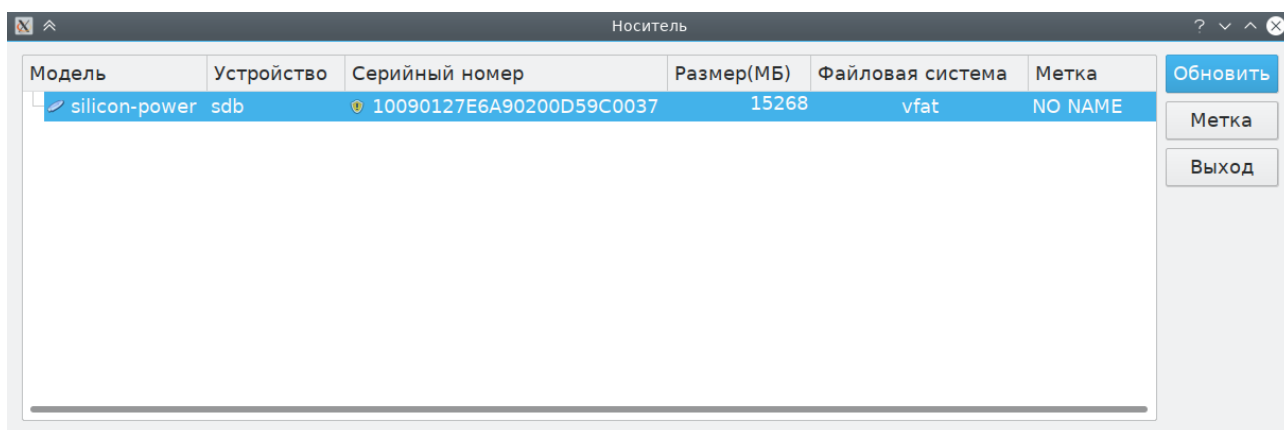


Рисунок 22 — Результат работы программы после обновления списка устройств с подключенным USB устройством

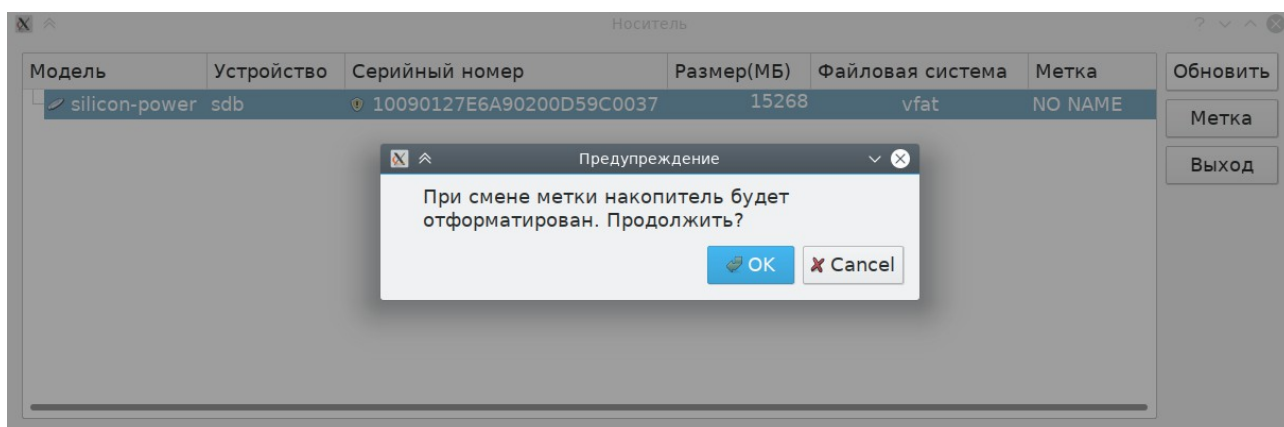


Рисунок 23 — Предупреждение при смене метки

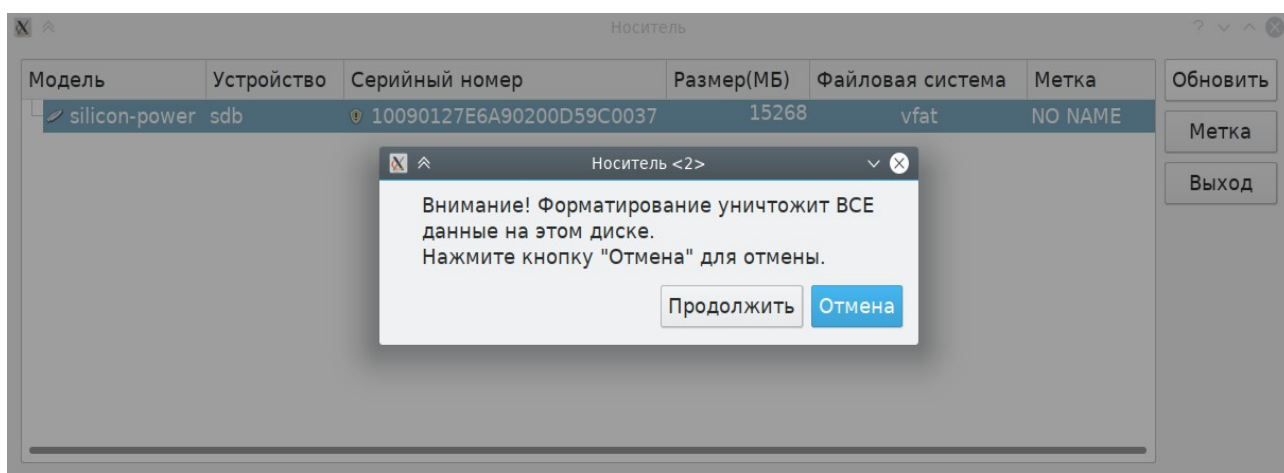


Рисунок 24 — Предупреждение при форматировании

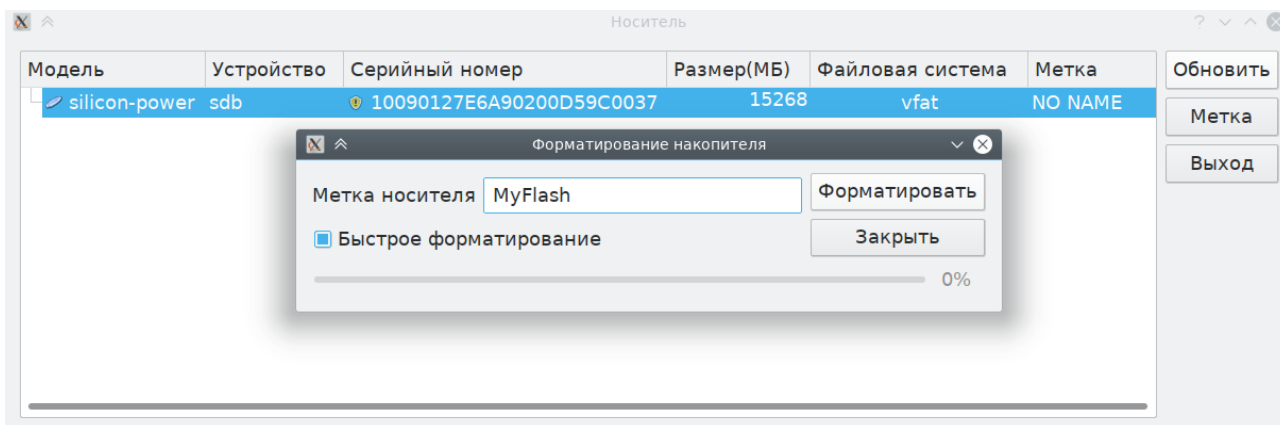


Рисунок 25 — Окно форматирования

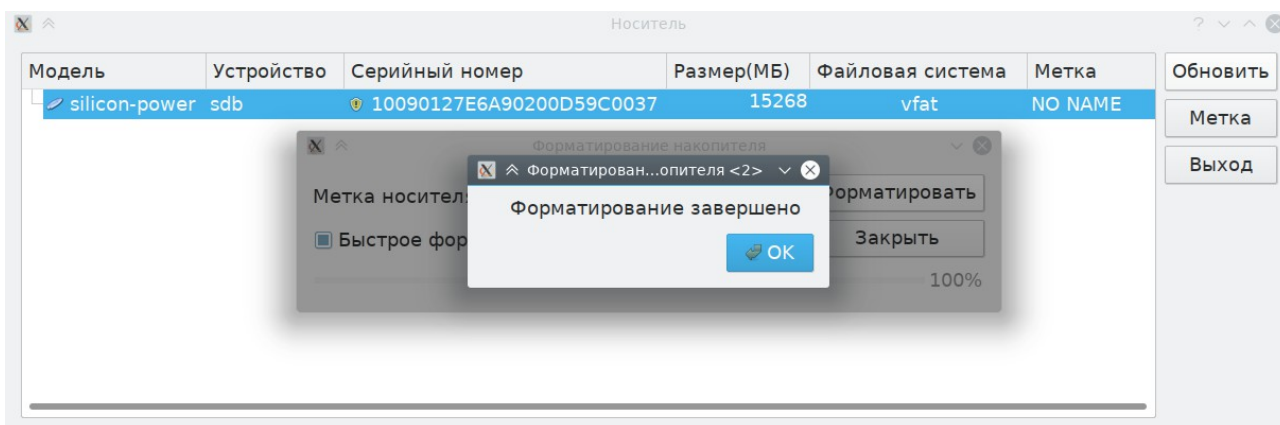


Рисунок 26 — Сообщение об успешном форматировании

После проведения процедуры смены метки на носителе отображаемая таблица в приложении была автоматически обновлена. В столбце «Серийный номер» появилось изображение, которое говорит о том, что хэш-сумма на устройстве верна. Результат представлен на рисунке 27.

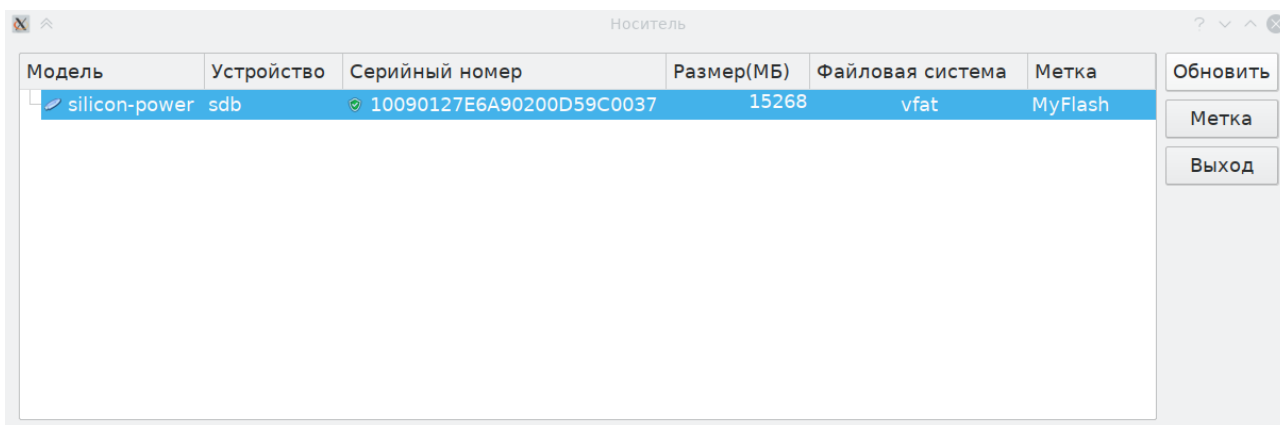


Рисунок 27 — Окно программы после форматирования и подписи USB устройства

Затем USB-устройство было отформатировано еще раз, но в без режима «Быстрое форматирование». В данном режиме устройство проверялось на битые сектора памяти, поэтому форматирование заняло около пятнадцати минут. При завершении форматирования было выдано сообщение об успешности операции. Результат представлен на рисунке 28 и 29.

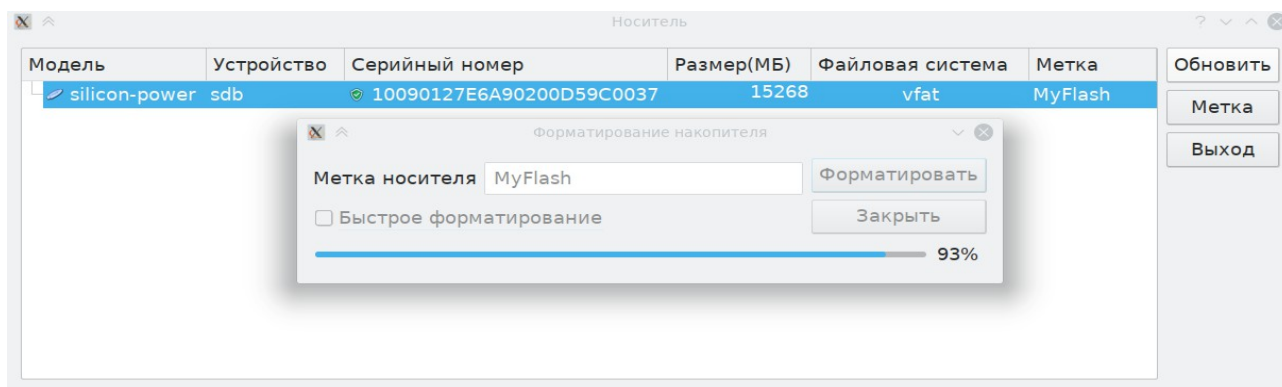


Рисунок 28 — Прогресс форматирования с проверкой битых секторов

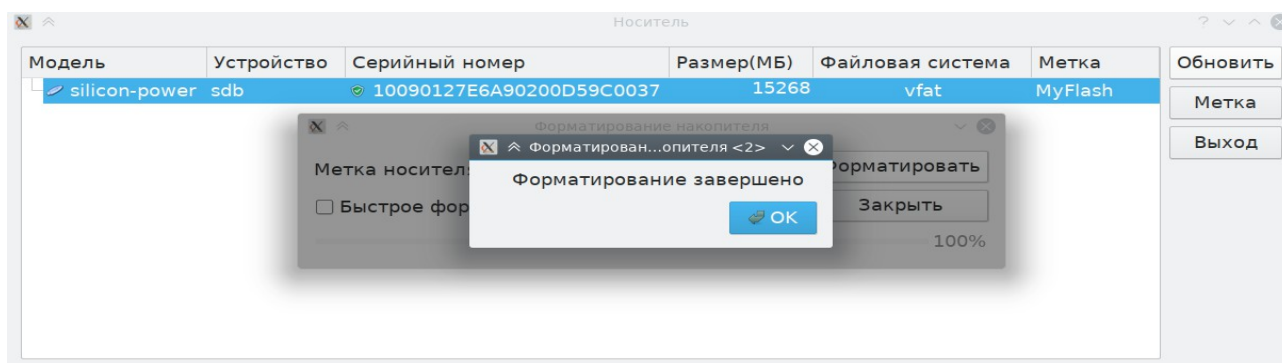


Рисунок 29 — Сообщение об успешном форматировании с проверкой битых секторов

4 Вредные психофизиологические факторы, влияющие на органы зрения специалиста по защите информации. Методы и средства физической культуры, снижающие их воздействие

4.1 Общие положения

Работа специалиста по защите информации напрямую связана с работой на компьютере. Продолжительная работа за компьютером вредит в первую очередь зрению. Эта проблема будет рассмотрена в данном разделе дипломного проекта.

4.2 Вредные психофизиологические факторы, влияющие на органы зрения специалиста по защите информации

Вредным производственным фактором называется такой производственный фактор, воздействие которого на работающего в определенных условиях приводит к заболеванию или снижению трудоспособности. Заболевания, возникающие под действием вредных производственных факторов, называются профессиональными. К вредным производственным факторам, влияющим на органы зрения специалиста по защите информации можно отнести:

- отсутствие или недостаток естественного света;
- недостаточная искусственная освещенность рабочей зоны;
- повышенная яркость света;
- повышенная контрастность;
- прямая и отраженная блескость;
- зрительное напряжение.

Работа на компьютере сопровождается постоянным и значительным напряжением функций зрительного анализатора. Одной из основных особенностей является иной принцип чтения информации, чем при обычном

чтении. При обычном чтении текст на бумаге, расположенный горизонтально на столе, считывается работником с наклоненной головой при падении светового потока на текст. При работе на ПК специалист по ИБ считывает текст, почти не наклоняя голову, глаза смотрят прямо или почти прямо вперед, текст формируется по другую сторону экрана, поэтому пользователь не считывает отраженный текст, а смотрит непосредственно на источник света, что вынуждает глаза и орган зрения в целом работать в несвойственном ему стрессовом режиме длительное время.

Расстройство органов зрения резко увеличивается при работе более четырех часов в день. Всемирная организация здравоохранения (ВОЗ) ввела понятие «компьютерный зрительный синдром» (КЗС), типовыми симптомами которого являются жжение в глазах, покраснение век и конъюнктивы, чувство инородного тела или песка под веками, боли в области глазниц и лба, затуманивание зрения, замедленная перефокусировка с ближних объектов на дальние.

Факторами, наиболее сильно влияющими на зрение, являются:

— недостатки способов создания изображения на экране монитора.

Эта группа факторов включает в себя:

- неоптимальные параметры схем развертки мониторов;
 - несовместимость параметров монитора и графического адаптера;
 - недостаточно высокое разрешение монитора, расфокусировка, несведение лучей и низкий уровень других его технических характеристик;
 - избыточная или недостаточная яркость изображения;
- непродуманная организация рабочего места, которая является причиной:
- наличия бликов на лицевой панели экрана;
 - отсутствия необходимого уровня освещенности рабочих мест;

- несоблюдения расстояния от глаз до экрана.

Блики относятся к факторам, которые очень сильно мешают воспринимать информацию с экрана монитора. Они заставляют неосознанно напрягать зрение, чтобы прочесть нужную информацию на экране. При этом глаза испытывают дополнительное напряжение, что приводит к быстрой утомляемости органа зрения.

Блики создает любой пучок света, отраженный экраном дисплея и попавший на оболочку глаза. Их источниками могут быть расположенные напротив монитора яркие поверхности, светлое оборудование, осветительные приборы, незашторенные окна. Блики тем заметнее и тем сильнее снижают контрастность изображения, чем выше коэффициент зеркального отражения экрана. В ряде случаев текст может стать фактически нечитаемым.

В процессе работы с персональным компьютером приходится работать с изображениями на экране монитора. Считывание текста, таблиц, графиков с экрана отличается от чтения той же информации с листа бумага по нескольким причинам:

- экран, являясь источником света, считается прибором активного контраста, в то время как при чтении с листа бумаги имеет дело отраженный текст, то есть с пассивным контрастом, который в малой степени зависит от интенсивности освещения и угла падения светового потока на бумагу;
- текст на бумаге является неизменным, а изображение на экране обновляется за секунду десятки раз. Достаточно низкая частота обновления вызывает мерцание изображений;
- специалист по ИБ зачастую проводит весь рабочий день, работая за компьютером, что является причиной длительной неподвижности глазных и внутриглазных мышц, в то время как они нуждаются в динамическом режиме работы. Это приводит к их ослаблению;

— длительная работа с персональным компьютером требует повышенной концентрации, что приводит к большим нагрузкам на зрительную систему пользователя. Развивается зрительное утомление (астенопия), которое способствует возникновению близорукости, головной боли, раздражительности, нервного напряжения и стресса.

Приведенные выше особенности изображений на экране дисплея, а также характера работы специалиста по защите информации в значительной мере влияют на степень утомляемости зрительного аппарата.

4.3 Методы и средства физической культуры, снижающие их воздействие

Для сохранения зрения и снижения воздействия вредных психофизиологических факторов специалисту по ИБ необходимы перерывы для снятия напряжения, в противном случае возможно развитие серьезных заболеваний. Специальная гимнастика для глаз поможет сохранить зрение.

Необходимо закрыть глаза и напрячь глазные мышцы на 4 секунды. Затем открыть глаза и расслабиться. Затем нужно посмотреть в окно вдаль около 6 секунд. Данное упражнение нужно повторить 4 или 5 раз.

Далее нужно перевести взгляд на переносицу и смотреть на нее примерно 4 секунды. Затем перевести взгляд на пейзаж за окном и смотреть туда около 6 секунд. Данное упражнение нужно повторить 4 или 5 раз.

После этого нужно посмотреть влево, при этом не поворачивая голову, зафиксировав глаза в этом положении примерно на 4 секунды. Затем нужно сделать то же самое, только смотря влево, вниз и вверх. Данное упражнение нужно повторить 3 или 4 раза.

Далее нужно выполнить повороты глазами в следующих направлениях: налево, вниз, направо, вверх, затем прямо вдаль в окно. Потом направо, вниз, налево, вверх, а дальше прямо вдаль в окно. Данное упражнение нужно повторить 3 или 4 раза.

После этого нужно моргнуть максимально быстро, отсчитав до 10, и потом закрыть глаза на несколько секунд. Затем нужно поморгать в течение минуты. Затем закрыть глаза на 2 или 3 секунды. После открыть их и посмотреть вдаль в окно. Повторить упражнение 2 или 3 раза.

Затем нужно рассмотреть любой хорошо видимый объект в течение 30 секунд. Потом перевести взгляд на самый удаленный предмет. Необходимо смотреть на него в течение 30 секунд. Потом вернуть взгляд на первый объект. Данное упражнение нужно повторить 6 раз.

После зарядки нужно дать глазам расслабиться, закрыв их. При этом не опускать голову вперед.

Выполнение подобных упражнений для глаз при работе за компьютером способствует их расслаблению и тренировке и снижает воздействие вредных психофизиологических факторов.

Заключение

В ходе выполнения дипломного проекта был произведен обзор методов защиты изделий вычислительной техники от несанкционированного доступа при помощи систем защиты информации и программно-аппаратных комплексов, а также был произведен обзор наиболее распространенных на сегодняшний день электронных идентификаторов. Были разработаны и описаны алгоритмы проектов, осуществляющих форматирование и создание на накопителе хеш-суммы.

В результате выполнения дипломного проекта было разработано программное обеспечение, позволяющее отформатировать отчуждаемый носитель информации, создать на нем уникальную цифровую последовательность и проверить ее при необходимости для дальнейшего использования носителя на вычислительной технике. Проведена экспериментальная проверка данной программы.

Данное программное обеспечение может быть применено без использования на изделии вычислительной техники дополнительных аппаратных модулей. В дальнейшем возможно внедрение в программу иных способов вычисления хеш-суммы. Возможна разработка дополнительного программного обеспечения, которое будет позволять отключать устройства в системе, если на них не найдена верная хеш-сумма. В данном случае для обмена информацией с изделием возможно обеспечить использование только специальных подписанных доверенных отчуждаемых устройств.

Таким образом, цель дипломного проекта достигнута, требования технического задания на выпускную квалифицированную работу были выполнены в полном объеме.

Протокол проверки на оригинальность представлен в приложении Е.

Список используемых источников

- 1 СЗИ от НСД Страж NT 4.0. [Электронный ресурс] // guardnt.ru: официальный сайт разработчиков. Режим доступа: https://www.guardnt.ru/gnt_40.html (свободный).
- 2 Обзор Secret Net Studio 8.1, защитные механизмы. [Электронный ресурс] // securitycode.ru: официальный сайт разработчиков. Режим доступа: https://www.securitycode.ru/company/press/obzor_secret_net_studio_8_1_chast_1_zashchitnye_mekhanizmy/ (свободный).
- 3 Сертифицированный аппаратно-программный модуль доверенной загрузки «Соболь». [Электронный ресурс] // securitycode.ru: официальный сайт разработчиков. Режим доступа: https://www.securitycode.ru/products/pak_sobol/ (свободный).
- 4 Сертификаты «Соболь». [Электронный ресурс] // securitycode.ru: официальный сайт разработчиков. Режим доступа: https://www.securitycode.ru/products/pak_sobol/certificates/ (свободный).
- 5 Аккорд-АМДЗ. [Электронный ресурс] // okbsapr.ru: официальный сайт разработчиков. Режим доступа: <https://www.okbsapr.ru/products/accord/accord-amdz/> (свободный).
- 6 Электронный идентификатор Guardant ID. [Электронный ресурс] // guardnt.ru: официальный сайт разработчиков. Режим доступа: <https://www.guardnt.ru/gid.html> (свободный).
- 7 Рутокен. [Электронный ресурс] // rutoken.ru: официальный сайт разработчиков. Режим доступа: <https://www.rutoken.ru/> (свободный).
- 8 Электронные идентификаторы iButton. [Электронный ресурс] // aladdin-rd.ru: официальный сайт разработчиков. Режим доступа: <https://www.aladdin-rd.ru/catalog/ibutton> (свободный).

9 Электронные USB-ключи и смарт-карты eToken. [Электронный ресурс] // aladdin-rd.ru: официальный сайт разработчиков. Режим доступа: <https://www.aladdin-rd.ru/catalog/etoken> (свободный).

10 Семейство продуктов JaCarta. [Электронный ресурс] // aladdin-rd.ru: официальный сайт разработчиков. Режим доступа: <https://www.aladdin-rd.ru/catalog/jacarta> (свободный).

11 USB-носители ESMART Token. [Электронный ресурс] // esmart.ru: официальный сайт разработчиков. Режим доступа: <https://esmart.ru/products/informatsionnaya-bezopasnost/usb-nositeli-esmart-token/> (свободный).

12 Электронные ключи для аутентификации ikey. [Электронный ресурс] // infobezpeka.com: официальный сайт разработчиков. Режим доступа: <http://www.infobezpeka.com/products/keyforauntification/?view=41> (свободный).

13 ГОСТ 28147-89. Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования. [Электронный ресурс] // Режим доступа: <http://docs.cntd.ru/document/gost-28147-89> (свободный).

Приложение А
(обязательное)
Руководство пользователя

Программный компонент «disktest»

Руководство пользователя

Оглавление

Аннотация.....	94
A.1 Назначение программы.....	95
A.1.1 Функциональное назначение программы.....	95
A.1.2 Эксплуатационное назначение программы.....	95
A.1.3 Состав функций.....	95
A.2 Условия выполнения программы.....	96
A.2.1 Минимальный состав аппаратных средств.....	96
A.2.2 Минимальный состав программных средств.....	96
A.2.3 Требования к персоналу (пользователю).....	96
A.3 Выполнение программы.....	97
A.3.1 Загрузка и запуск программы.....	97
A.3.2 Выполнение программы.....	97
A.3.3 Завершение работы программы.....	98
A.4 Сообщения оператору.....	99

Аннотация

В данном документе приведено руководство пользователя по применению и эксплуатации программного компонента «disktest», предназначенного форматирования отчуждаемого носителя информации, чтения, записи, получения и проверки контрольной суммы.

А.1 Назначение программы

А.1.1 Функциональное назначение программы

Функциональным назначением программы «disktest» является создание доверенного отчуждаемого носителя информации для последующего его использования при обмене информацией с вычислительной техникой.

А.1.2 Эксплуатационное назначение программы

Программа «disktest» разработана для применения в учебных целях. Пользователям программы может быть любой человек, обладающий базовыми навыками использования персонального компьютера..

А.1.3 Состав функций

Программа обеспечивает возможность выполнения перечисленных ниже функций:

- функция определения подключенных к вычислительному средству носителей информации;
- функция определения объема памяти, файловой системы, метки накопителя;
- функция форматирования с созданием на устройстве файловой системы FAT32;
- функция чтения и проверки контрольной суммы на устройстве;
- функция генерации и записи контрольной суммы на отчуждаемый носитель.

А.2 Условия выполнения программы

А.2.1 Минимальный состав аппаратных средств

Минимальный состав используемых технических (аппаратных) средств:

- ПК с процессором с частотой не менее 1,3 ГГц;
- оперативная память объемом не менее 4 Гб;
- жесткий диск объемом 320 Гб и выше.

А.2.2 Минимальный состав программных средств

Для корректной работы программы необходимо наличие библиотек Qt Creator в системе.

А.2.3 Требования к пользователю

Минимальные знания о работе с операционными системами Linux, базовые навыки владения вычислительной техникой.

А.3 Выполнение программы

А.3.1 Загрузка и запуск программы

С помощью диспетчера файлов или консоли необходимо запустить исполняемый файл. Для корректной работы программы необходимо осуществлять ее запуск с правами суперпользователя.

А.3.2 Выполнение программы

При появлении графического интерфейса программы необходимо нажать на кнопку «Обновить» для определения подключенных накопителей, устройство должно быть заранее подключено к вычислительному устройству. После появления списка устройств в таблице графического интерфейса необходимо нажать кнопку «Метка», чтобы создать и записать на устройство контрольную сумму. После нажатия кнопки «Метка» пользователю будет выдано сообщение о том, что накопитель при смене метки будет отформатирован. Для продолжения процедуры в появившемся окне необходимо нажать кнопку «ОК», для отмены необходимо нажать «Cancel». Далее пользователю в появившемся окне будет выдано сообщение о том, что форматирование уничтожит все файлы на накопителе. Чтобы продолжить, необходимо нажать на кнопку «Продолжить», в ином случае необходимо нажать кнопку «Отмена».

Далее пользователю будет показано окно, в котором есть поле ввода текста «Метка носителя», кнопка «Форматировать», кнопка «Закрыть», шкала прогресса, которая будет неактивна до начала выполнения процедуры форматирования, опция «Быстрое форматирование». В поле ввода текста «Метка носителя» изначально будет введена метка «Flash». Введенная метка пользователем должна соответствовать уже заданному в поле шаблону. Таким образом, пользователю позволено добавлять цифры в конец уже введенной метки, но необходимо, чтобы общая длина метки не превышала 11 символов.

Например, метка, введенная пользователем может быть «Flash32». После ввода метки, если пользователь хочет провести форматирование в режиме проверки битых секторов памяти на устройстве, он должен снять опцию «Быстрое форматирование» кликом по квадрату рядом с соответствующей надписью в графическом интерфейсе или по самой надписи. После этого, чтобы запустить форматирование, необходимо нажать клавишу «Форматировать». Чтобы прервать процедуру необходимо нажать кнопку «Закрыть». В случае, если режим быстрого форматирования был отключен, форматирование может занять продолжительное время, в зависимости от объема памяти форматируемого накопителя. При успешном форматировании пользователю будет выдано соответствующее сообщение. При нажатии кнопки «ОК» окно процедуры форматирования будет закрыто и пользователь вернется к начальному окну программы со списком устройств.

Чтобы определить пользователю верна ли хеш-сумма на накопителе, необходимо, после выполнения обновления таблицы устройств при помощи клавиши «Обновить», обратить внимание на столбец «Серийный номер» в таблице. Рядом с серийным номером устройства будет изображена соответствующая иконка. В случае, если контрольная сумма на накопителе верна, на иконке будет изображен знак галочки на зеленом фоне щита. В случае, если произошла какая-либо ошибка или контрольная сумма на накопителе не найдена или неверна, то на иконке будет изображен восклицательный знак на желтом фоне щита.

А.3.3 Завершение работы программы

Завершение программы, осуществляется с помощью кнопки «Выход».

А.4 Сообщения пользователю

В процессе работы пользователю могут быть выданы сообщения об успешном завершении работы какого-либо процесса или же о возникшей ошибке. В обоих случаях, в появившемся окне будет сообщение с конкретным описанием результата завершения процесса, благодаря которому пользователь получит исчерпывающую информацию о работе программы.

Приложение Б

(обязательное)

Общий алгоритм программы

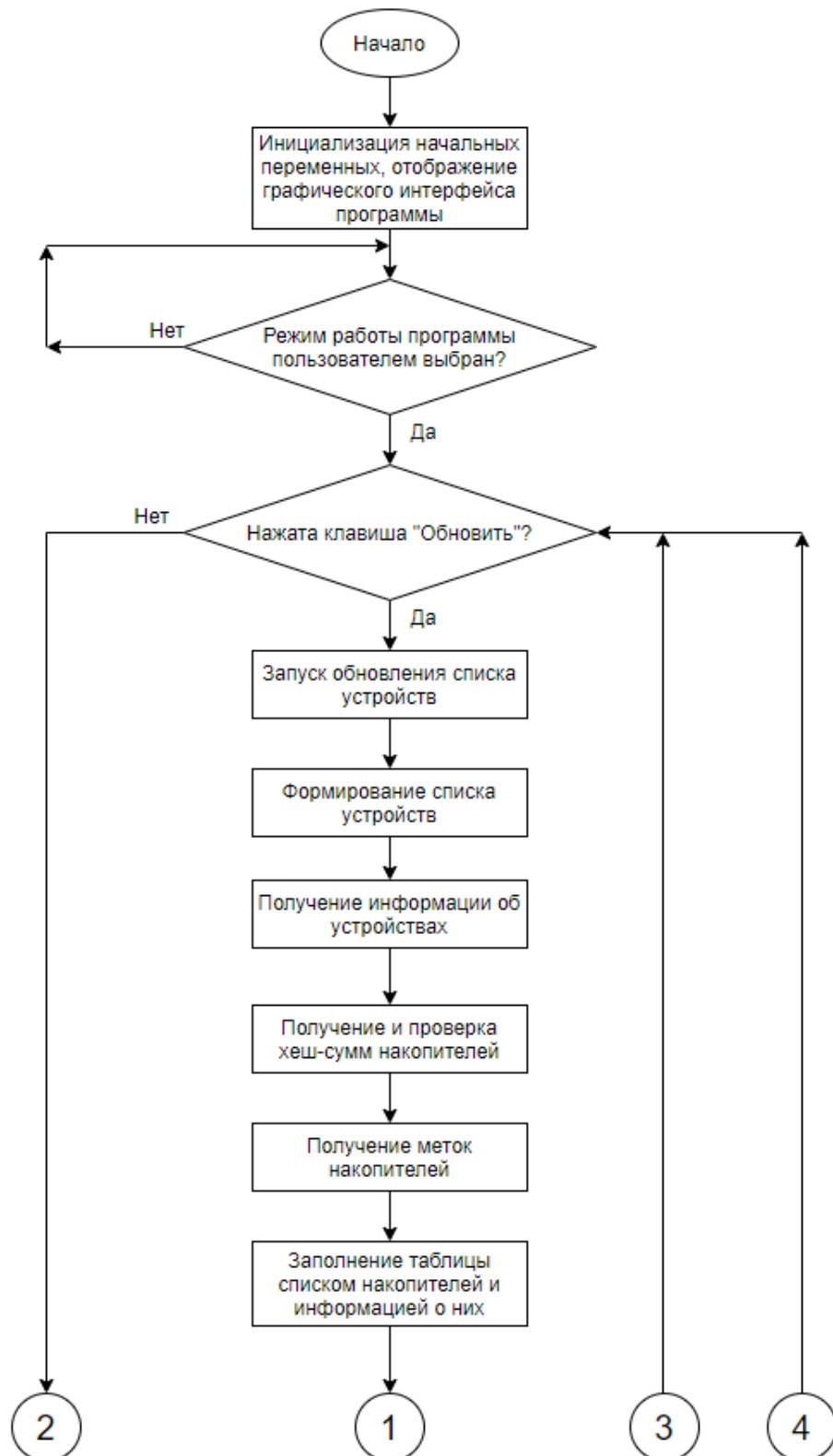


Рисунок Б.1 — Общий алгоритм программы, часть 1

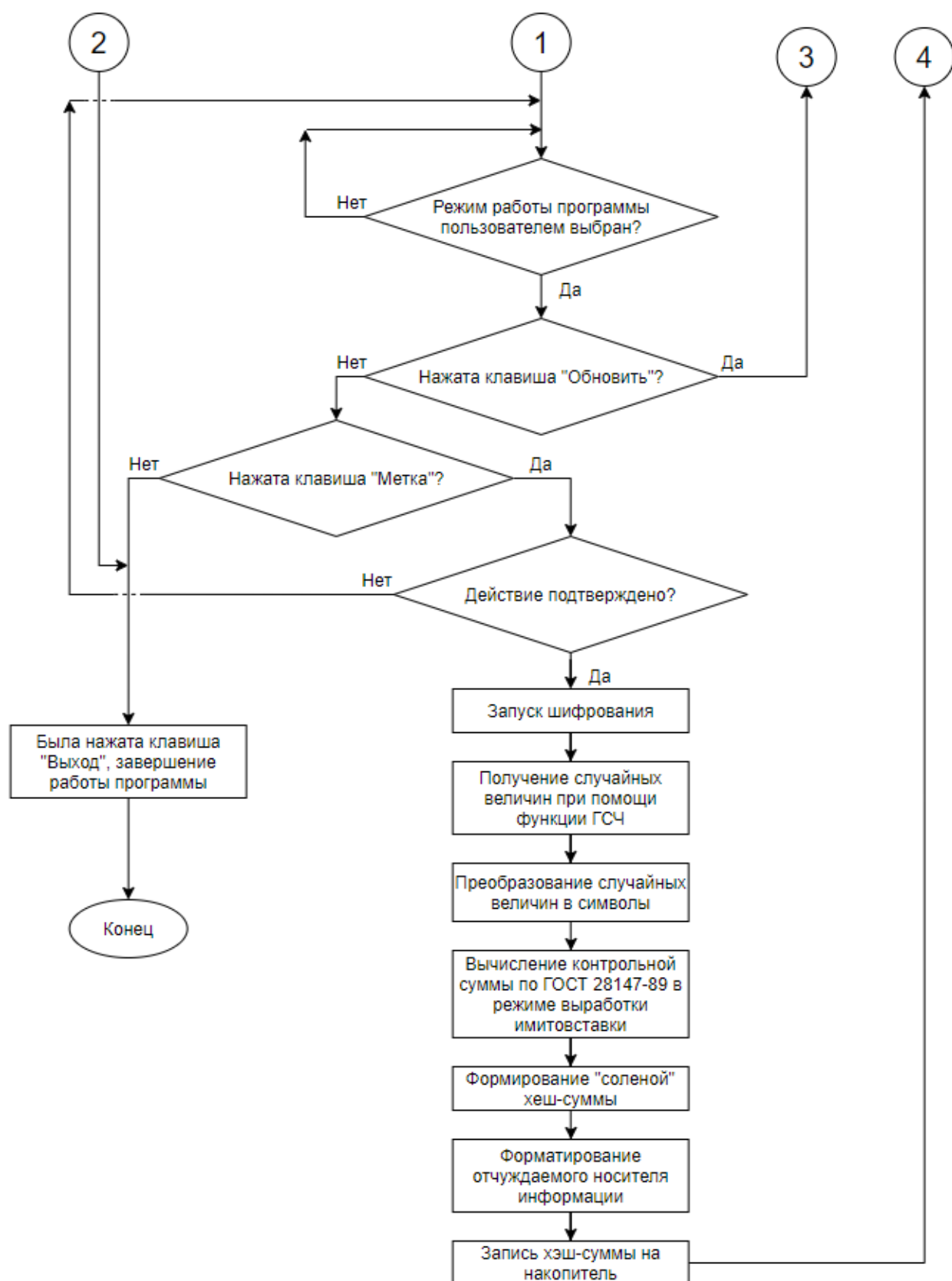


Рисунок Б.2 — Общий алгоритм программы, часть 2

Приложение В

(обязательное)

Алгоритм проекта, отвечающего за форматирование



Рисунок В.1 — Алгоритм проекта, отвечающего за форматирование

Приложение Г

(обязательное)

Код проекта, отвечающего за формирование графического интерфейса приложения, определение подключенных устройств, получение хеш-суммы с использованием ГОСТ 28147-89 в режиме выработки имитовставки, проверку, чтение, запись хеш-суммы на накопитель.

Код заголовочного файла disktest.h:

```
#ifndef DISKTEST_H
#define DISKTEST_H

#include <QDialog>
#include <QtCore>
#include <QTreeWidgetItem>
#include <QProgressDialog>
#include <QList>
#include <QTimer>

#define VNI_FLASH 1
#define VNI_CDROM 2
#define VNI_HARDDISK 3

#define EXT3_FEATURE_COMPAT_HAS_JOURNAL 0x0004
#define VOLNAMSZ 16

namespace Ui
{
class disktest;
}

typedef struct
{
    QByteArray fileDev;          // имя файла устройства
```

```

    QByteArray mountPoint;    // точка монтирования
    QByteArray vendor;        // производитель
    QByteArray model;         // модель
    QByteArray label;         // метка
    QByteArray serialNumber;  // серийный номер
    QByteArray hash;          // хэш
    off64_t devSize;          // размер устройства
    int fs_type;               // тип ФС
    int removable;            // признак съёмного носителя
    int dev_type;              // тип устройства
    bool reg;                  // признак регистрации носителя
} TStDevInfo;

```

```

class QTWIDev : public QTreeWidgetItem
{
public:
    QTWIDev(QTreeWidgetItem *view, int type = Type) :
    QTreeWidgetItem(view, type){}
    QTWIDev(QTreeWidgetItem *parent, int type = Type) :
    QTreeWidgetItem(parent, type){}

    TStDevInfo m_devInfo;
};

```

```

unsigned long from16(unsigned char * buf, int size);

```

```

//! Возможные классы накопителей
enum StorageClass {
    StorageClassLocal
};

```

```

// параметры файловой системы NTFS
struct ntfs_param
{

```

```

        unsigned long sec_size;           // размер сектора в байтах
        unsigned long clust_size;         // размер кластера в байтах
        unsigned long file_size;          // размер файла в байтах
        unsigned long mft_addr;           // адрес записи mft
        unsigned long mft_mirror_addr;    // адрес копии записи mft
mft_mirror
        unsigned long addr_upseq;         // адрес последовательности
обновления
        int size_upmass;                  // размер массива обновления
        unsigned long attr_offset;        // смещение атрибута
        int label_offset;                 // смещение метки
        unsigned long label_size;         // размер метки
        QString label;                   // метка
};

struct ext2_super_block
{
    char s_dummy0[56];
    unsigned char s_magic[2];
    char s_dummy1[34];
    unsigned int s_feature_compat;
    char s_dummy3[24];
    char s_volume_name[VOLNAMSZ];
    char s_last_mounted[64];
    char s_dummy2[824];
}__attribute__((packed));

static const char * const fs_types[] = {"msdos", "msdos", "vfat",
"ext2", "ext3", "ext4", "ntfs", "отсутствует"};

enum
{
    FAT12 = 0,
    FAT16,

```

```

        FAT32,
        EXT2,
        EXT3,
        EXT4,
        NTFS,
        NOT_SUPPORTED
};

class disktest : public QDialog {
    Q_OBJECT

public:
    disktest(QWidget *parent = 0);
    ~disktest();

    void reloadLocal();
    void init();
    void fillTree(QTreeWidgetItem* root, QList<TStDevInfo>
partList, StorageClass storageClass = StorageClassLocal);
    static int checkSerialHash(TStDevInfo devInfo);
    static QList<TStDevInfo> mkPartitionList(QList<TStDevInfo>&
dev_info_list);
    static QByteArray readHash(TStDevInfo devInfo);
    static int getLabel(TStDevInfo &devInfo, bool
partition=false);
    static int getStorageUSBSerial(TStDevInfo *dev_info);

private:
    Ui::disktest *ui;
    char *t_buf_write,*t_buf_read;
    QTimer m_timer;

    int test_local(QTWIDev* dev, bool full_test);
    void write_hash(int fd, const char *hash);

```

```
private slots:
```

```
void on_update_button_clicked();
void on_formatting_button_clicked();
void formatting_local_drive();

void on_m_metka_button_clicked();
void on_treeWidget_currentItemChanged(QTreeWidgetItem
*current, QTreeWidgetItem *previous);
char* hashSerialNumber(char *serialNumber);
char mksalt(int seed);
int VYR_GSCH(unsigned char* buf, unsigned int count);
```

```
protected:
```

```
static TStDevInfo getDevInfo(const char *dev, bool
partition=false);
static ntfs_param * getNTFSParams(char * dev);
static int getSCSIDevInfo(TStDevInfo *dev_info, bool
partition=false );
};
#endif // DISKTEST_H
```

Код файла main.cpp:

```
//file Файл содержит реализацию основной функции приложения
#include <QtGui/QApplication>
#include "disktest.h"
#include <syslog.h>

//! главная функция приложения.
/**
Главная функция приложения. Создает объекты приложения и
инициализирует его.
\param argc количество аргументов.
\param argv указатель на аргументы.
```

```

        \return код возврата.
*/
int main(int argc, char *argv[])
{
    QTextCodec::setCodecForCStrings(QTextCodec::codecForName("UTF-
8"));
    openlog("Тестирование устройств", LOG_PID, LOG_USER);
    QApplication a(argc, argv);
    disktest w;
    w.show();
    return a.exec();
}

```

Код файла disktest.cpp:

//! \file src/disktest.cpp реализация функций главного окна приложения.

```

#include "disktest.h"
#include "ui_disktest.h"
#include <syslog.h>
#include <fcntl.h>
#include <src/giz.c>

#include "qmessagebox.h"
#include "unistd.h"

#include "sys/ioctl.h"
#include "sys/mount.h"
#include "sys/stat.h"

#include <linux/cdrom.h>

#include <sys/ioctl.h>
#include <scsi/scsi.h>
#include <scsi/sg.h>

```

```

#define aName          "MyProject"
#define TBUF_SIZE      65536          //64KB (делать кратным
patterns - 4b)

#define LOGERRm LOG_ERR, "Ошибка %s:%d, %m", __FILE__, __LINE__

#define HASH_SIZE 13
#define HASH_OFFSET 0x1B0

#define EXT2_SUPER_MAGIC 0xEF53
#define BUFSIZE 16*1024

//! Конструктор класса disktest.
/**
Создает объекты элементов управления и инициализирует их.
\param parent указать на родительское окно.
*/
disktest::disktest(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::disktest)
{
    ui->setupUi(this);

    setWindowTitle(aName);

    setWindowIcon(QIcon(":/disktest.png"));

    //Выделяем память под тестовый буфер
    t_buf_write = (char*)malloc(TBUF_SIZE);
    t_buf_read = (char*)malloc(TBUF_SIZE);

    ui->formatting_button->hide();

```

```

        init();
    }

    //! Деструктор объекта
    disktest::~disktest()
    {
        //Освобождаем тестовый буфер
        free(t_buf_write);
        free(t_buf_read);

        syslog(LOG_INFO, "Приложение завершено");
        closelog();

        delete ui;
    }

    //! Инициализация буферов чтения/записи
    void disktest::init()
    {
        unsigned char patterns[] = {0xaa, 0x55, 0xff, 0x00};

        memset(t_buf_write, 0, TBUF_SIZE);
        memset(t_buf_read, 0, TBUF_SIZE);

        for(unsigned int bi = 0; bi < TBUF_SIZE; bi +=
sizeof(patterns))
            memcpy(t_buf_write + bi, patterns, sizeof(patterns));
    }

    /**
     * @brief Заполнение дерева накопителей
     * @param root корневой элемент
     * @param partList список с элементами дерева
     * @param storageClass тип элементов

```

```

*/
void disktest::fillTree(QTreeWidgetItem *root, QList<TStDevInfo>
partList, StorageClass storageClass)
{
    QTreeWidgetItem* pitem = 0;
    QTreeWidgetItem* item = 0;
    QByteArray pass_dev;

    for(int i = 0; i < partList.count(); i++)
    {
        if(!pass_dev.isEmpty() &&
partList.at(i).fileDev.startsWith(pass_dev))
            continue;

        if(partList.at(i).fileDev.right(1).toInt() != 0 && pitem)
        {
            item = new QTreeWidgetItem(pitem);
            item->setText(0, "Раздел");
            pitem->setExpanded(true);

            item->setIcon(0, QIcon(":/images/kcmlpartitions.png"));
        }
        else
        {
            item = new QTreeWidgetItem(root);
            pitem = item;
            item->setText(0, partList.at(i).model);

            if(partList.at(i).removable)
                item->setIcon(0, QIcon(":/images/media-usb-
flash.png"));
            else
                item->setIcon(0, QIcon(":/images/drive-

```

```

harddisk.png"));
    }

    if(storageClass == StorageClassLocal)
        item->setText(1,
QFileInfo(partList.at(i).fileDev).fileName());

    if(!partList.at(i).serialNumber.isEmpty())
    {

        int res = checkSerialHash(partList[i]);

        switch(res)
        {
        case 0:
            syslog(LOG_INFO, "%s", QString("Проверка
устройства %1 пройдена
успешно!").arg(partList.at(i).fileDev.data()).toAscii().data());
            item->setIcon(2, QIcon(":/images/vni_check.png"));
            break;
        case 1:
            syslog(LOG_INFO, "%s", QString("Ошибка! Неверный
криптографический ключ на устройстве
%1!").arg(partList.at(i).fileDev.data()).toAscii().data());
            item->setIcon(2, QIcon(":/images/vni_warn.png"));
            break;
        case 2:
            item->setIcon(2, QIcon(":/images/vni_warn.png"));
            break;
            item->setIcon(2, QIcon(":/images/vni_warn.png"));
        default:
            break;
        }
    }

```

```

    }
    else if(partList[i].removable)
        item->setIcon(2, QIcon(":/images/vni_warn.png"));

    item->setText(2, partList.at(i).serialNumber);

    item->setText(3,
QVariant((qlonglong)partList.at(i).devSize/1024/1024).toString());
    item->setTextAlignment(3, Qt::AlignRight);
    item->setText(4, fs_types[partList.at(i).fs_type]);
    item->setTextAlignment(4, Qt::AlignCenter);
    item->setText(5, partList.at(i).label);

    item->setData(0, Qt::UserRole, storageClass);

    item->m_devInfo = partList.at(i);
}

ui->treeWidget->resizeColumnToContents(0);
ui->treeWidget->resizeColumnToContents(1);
ui->treeWidget->resizeColumnToContents(2);
ui->treeWidget->resizeColumnToContents(3);
ui->treeWidget->resizeColumnToContents(4);
ui->treeWidget->resizeColumnToContents(5);

if(ui->treeWidget->topLevelItemCount() > 0)
{
    ui->treeWidget->itemAt(0, 0)->setSelected(true);
    ui->treeWidget->setCurrentItem(ui->treeWidget->itemAt(0,
0));
}
}

//! Обновление списка на локальной машине

```

```

void disktest::reloadLocal()
{
    QList<TStDevInfo> partList = mkPartitionList(partList);

    fillTree(ui->treeWidget->invisibleRootItem(), partList);
}

//! Запуск форматирования накопителя на локальной машине
void disktest::formatting_local_drive()
{
    const char* command = CHROOT"/fdrive/fdrive";
    QTWIDev* twi = (QTWIDev*)ui->treeWidget->currentItem();

    if(access(command, X_OK))
    {
        QMessageBox::critical(this, "Ошибка", QString("not found
%1").arg(command));
        this->update();
        return;
    }

    if(!twi)
        return ;

    QTWIDev* root_parent = twi;
    while(root_parent->parent())
        root_parent = (QTWIDev*)root_parent->parent();

    char *hash_buff = NULL;

    if(twi->parent())
        hash_buff = hashSerialNumber(((QTWIDev *)root_parent)-
>m_devInfo.serialNumber.data());
    else

```

```

        hash_buff = hashSerialNumber(twi-
>m_devInfo.serialNumber.data());

        QProcess process;
        process.start(command, QStringList() << twi->m_devInfo.fileDev
<< "1");

        if(!process.waitForStarted()) {
            const char *msg = "Не удалось запустить
приложение \"Форматирование накопителя\"";
            syslog(LOG_INFO, msg);
            QMessageBox::critical(this, "Ошибка", msg);

            return ;
        }

        process.waitForFinished(-1);

        if((process.exitStatus() != QProcess::NormalExit) ||
(process.exitCode()))
        {
            syslog(LOG_ERR, "Ошибка, %s:%d, Некорректное завершение
приложения %s: %m", __FILE__, __LINE__, "Форматирование
накопителя");
            QMessageBox::critical(this, "Ошибка", "Некорректное
завершение приложения " + QString("Форматирование накопителя"));
            return ;
        }

        int repart_fd = ::open(root_parent->m_devInfo.fileDev.data(),
O_RDWR);
        if(repart_fd != -1)
        {

```

```

        write_hash(repart_fd, hash_buff);

        if(ioctl(repart_fd, BLKRRPART, 0) == -1)
        {
            syslog(LOG_ERR, "Ошибка, %s:%d, ioctl(repart_fd,
BLKRRPART, 0) %s: %m", __FILE__, __LINE__, twi-
>m_devInfo.fileDev.data());
            QMessageBox::critical(this, "Ошибка", "Ошибка
обновления таблицы разделов");
        }
        ::close(repart_fd);
    }
}

//! Шифрование пароля
/**
 * \param password пароль
 * \return строка зашифрованного пароля
 */
char* disktest::hashSerialNumber(char *serialNumber)
{
    char thesalt[3];
    unsigned char mas[2];
    VYR_GSCH(mas, 2);
    thesalt[0] = mksalt(mas[0]);
    thesalt[1] = mksalt(mas[1]);
    thesalt[2] = '\0';

    char* crypt_password = passwordhash(serialNumber, thesalt);
    return crypt_password;
}

int disktest::VYR_GSCH(unsigned char* buf, unsigned int count)

```

```

{
    int i;
    srand(time(NULL));
    for(i = 0; i<count; i++)
        buf[i] = rand();

    return 0;
}

//! Возвращает символ из набора a-z, A-Z, 0-9, /, .
/**
 * \param seed случайный инициализатор
 * \return символ
 */
char disktest::mksalt(int seed)
{
    int num = seed % 64;

    if (num < 26)
        return 'a' + num;
    else if (num < 52)
        return 'A' + (num - 26);
    else if (num < 62)
        return '0' + (num - 52);
    else if (num == 63)
        return '.';
    else
        return '/';
}

/**
 * @brief Запись криптографической свёртки на накопитель
 * @param fd дескриптор накопителя
 * @param hash свёртка

```

```

*/
void disktest::write_hash(int fd, const char *hash)
{
    if(!hash)
        return ;

    ::lseek(fd, HASH_OFFSET, SEEK_SET);
    ssize_t wr = ::write(fd, hash, HASH_SIZE);
    if(wr < 0) {
        int err = errno;
        syslog(LOG_INFO, "Ошибка записи криптографической метки на
носитель: %m (%d)", err);
    }
}

//! Смена метки накопителя
void disktest::on_m_metka_button_clicked()
{
    if(QMessageBox::question(this, "Предупреждение", "При смене
метки накопитель будет отформатирован. Продолжить?",
                                QMessageBox::Ok, QMessageBox::Cancel)
== QMessageBox::Ok)
        on_formatting_button_clicked();
}

//! Активирует кнопки главного окна приложения при выборе
накопителя из списка.
/**
    \param current указатель на текущий элемент списка
    \param prev указатель на предыдущий выбранный элемент списка
*/
void disktest::on_treeWidget_currentItemChanged(QTreeWidgetItem
*current, QTreeWidgetItem *prev)
{

```

```

Q_UNUSED(prev);

if(current)
{
    //ui->fast_test_button->setDisabled(false);
    //ui->full_test_button->setDisabled(false);
    ui->formatting_button->setDisabled(false);
    ui->m_metka_button->setDisabled(false);
}
else
{
    //ui->fast_test_button->setDisabled(true);
    //ui->full_test_button->setDisabled(true);
    ui->formatting_button->setDisabled(true);
    ui->m_metka_button->setDisabled(true);
}
}

/**
 * @brief проверка хэша
 * @param devInfo информация об устройстве
 * @return 0 в случае успеха
 */
int disktest::checkSerialHash(TStDevInfo devInfo)
{
    if (!devInfo.serialNumber.isEmpty())
    {
        QByteArray hash = devInfo.hash;

        QByteArray serialNumber = devInfo.serialNumber;

        char* hash_from_serial = passwordhash(serialNumber.data(),
hash.data());

```

```

        if(hash_from_serial) {
            if(strncmp(hash_from_serial, hash.data(), HASH_SIZE))
{
                syslog(LOG_INFO, "%s", QString("Ошибка! Неверный
криптографический ключ На устройстве
%1!").arg(devInfo.fileDev.data()).toAscii().data());
                return 1;
            } else {
                syslog(LOG_INFO, "%s", QString("Проверка
устройства %1 пройдена
успешно!").arg(devInfo.fileDev.data()).toAscii().data());
                return 0;
            }
        } else {
            //syslog(LOG_INFO, "%s", QString("Не удалось выполнить
проверку криптографического ключа на устройстве
%1").arg(devInfo.fileDev.data()).toAscii().data());
            return 2;
        }
    }

    return 1;
}

//! Обновляет список накопителей.
void disktest::on_update_button_clicked()
{
    ui->treeWidget->clear();

    init();
    reloadLocal();

    ui->update_button->setFocus();
}

```

```

}

//! Функция вызывает программу форматирования
void disktest::on_formatting_button_clicked()
{
    QTWIDev* twi = (QTWIDev*)ui->treeWidget->currentItem();
    if(!twi)
        return ;

    if (QMessageBox::warning(this,
                             aName,
                             "Внимание! Форматирование уничтожит
ВСЕ данные на этом диске.\nНажмите кнопку \"Отмена\" для отмены.",
                             "&Продолжить", "&Отмена", NULL, 1))
        return ;

    int twi_class = twi->data(0, Qt::UserRole).toInt();
    if(twi_class == StorageClassLocal)
        formatting_local_drive();

    on_update_button_clicked();
}

/**
 * @brief чтение хэша
 * @param devInfo информация об устройстве
 * @return хэш
 */
QByteArray disktest::readHash(TStDevInfo devInfo)
{
    QByteArray ret;

    int file_fd = ::open(devInfo.fileDev.data(), O_RDONLY);
    char hash_buff[HASH_SIZE] = {0,};

```

```

        if(file_fd < 0)
            return ret;

        ::lseek(file_fd, HASH_OFFSET, SEEK_SET);
        ssize_t bytes = read(file_fd, &hash_buff, HASH_SIZE);
        ::close(file_fd);
        if(bytes != HASH_SIZE)
            return ret;

        ret = QByteArray(hash_buff, HASH_SIZE);

        return ret;
    }

    //! функция получения метки носителя
    /**
        \param devInfo структура параметров устройства, в которую
        записывается метка
        \param partition флаг учета разделов, partition=true -
        возвращает метку SCSI устройства с учетом разделов, иначе -
        возвращает метку внешних устройств без учета разделов
        \return код ошибки
    */
    int disktest::getLabel(TStDevInfo &devInfo, bool partition)
    {
        int fd;

        int bufsize=BUFSIZE;
        unsigned char buff[bufsize];
        memset(buff, 0, sizeof(buff));
        if( -1 != (fd = ::open( devInfo.fileDev.data(), O_RDONLY)) )
        {
            devInfo.devSize = ::lseek64(fd, 0, SEEK_END);
            ::lseek( fd, 0, SEEK_SET );

```

```

        //Читаем mbr для проверки файловой системы
        int n = ::read( fd, buff,  bufsize);

        ::close(fd);
        if(n==-1 && -1 != (fd = ::open( devInfo.fileDev.data(),
O_RDONLY | O_NONBLOCK)) )
        {
            n = ::read( fd, buff,  bufsize);
            ::close(fd);
        }
        if( bufsize != n )
        {
            syslog(LOG_WARNING, "Ошибка, %s:%d, чтение %s: %m",
__FILE__, __LINE__, devInfo.fileDev.data());
            QMessageBox::warning( 0, "Носитель", "Ошибка чтения
накопителя");
            return -1;
        }
    }
    else
    {
        syslog(LOG_WARNING, "Ошибка, %s:%d, открытие %s: %m",
__FILE__, __LINE__, devInfo.fileDev.data() );
        return -1;
    }

    struct ext2_super_block* sbb = (struct ext2_super_block*)
(buff+1024);

    if(devInfo.dev_type != VNI_CDROM)
    {
        if( 0 == memcmp("FAT12", buff+54, 5) )
            devInfo.fs_type = FAT12;
        else if( 0 == memcmp("FAT16", buff+54, 5) )

```

```

        devInfo.fs_type = FAT16;
    else if( 0 == memcmp("FAT32", buff+82, 5) )
        devInfo.fs_type = FAT32;

    else if( sbb->s_magic[0] + 256*sbb->s_magic[1] ==
EXT2_SUPER_MAGIC )
    {
        if(sbb->s_feature_compat &
EXT3_FEATURE_COMPAT_HAS_JOURNAL)
            devInfo.fs_type = EXT3;
        else
            devInfo.fs_type = EXT2;
    }
    else if( buff[0] == 0xEB &&
            buff[1] == 0x52 &&
            buff[2] == 0x90 &&
            buff[3] == 0x4E && // N
            buff[4] == 0x54 && // T
            buff[5] == 0x46 && // F
            buff[6] == 0x53 ) // S
        devInfo.fs_type = NTFS;
    }

    if(!partition)
    {
        syslog(LOG_WARNING, "Неверная файловая система на
накопителе %s", devInfo.fileDev.data());
        return -1;
    }

    switch( devInfo.fs_type )
    {
    case FAT12:
    case FAT16:

```

```

        devInfo.label = QByteArray((const char*)buff+43, 11);
        break;
case FAT32:
        devInfo.label = QByteArray((const char*)buff+71, 11);
        break;
case EXT2:
case EXT3:
case EXT4:
        devInfo.label = QByteArray(sbb->s_volume_name, 16);
        break;
case NTFS:
        struct ntfs_param *
ntfs_prm=getNTFSParams(devInfo.fileDev.data());
        if(ntfs_prm!=NULL)
            devInfo.label = ntfs_prm->label.toAscii();
        break;
    }

    devInfo.reg = (devInfo.label[0]=='\0') && (devInfo.label[1]!
='\0');
    if(devInfo.reg)
        devInfo.label = devInfo.label.mid(1);

    if(devInfo.fs_type!=NTFS)
        devInfo.label = QTextCodec::codecForName("KOI8-R")->
toUnicode(devInfo.label).toAscii();
    return 0;
}

//! функция получения параметров файловой системы NTFS
/**
    \param dev имя файла устройства
    \return параметры файловой системы NTFS
*/

```

```

ntfs_param * disktest::getNTFSParams(char * dev)
{
    int boot_size=512;

    int fd;
    if( -1 == (fd = ::open( dev, O_RDONLY )) )
        return NULL;

    unsigned char * buff = (unsigned char *)malloc(boot_size);
    memset(buff, 0, boot_size);

    if( boot_size != ::read( fd, buff, boot_size) )
    {
        syslog(LOG_WARNING, "Ошибка, %s:%d, открытие %s: %m",
__FILE__, __LINE__, dev );
        free(buff);
        QMessageBox::warning( 0, "Носитель", "Ошибка открытия
накопителя");
        return NULL;
    }

    ntfs_param * ntfs_prm = new ntfs_param();

    ntfs_prm->sec_size = from16(buff+0xb, 2);
    ntfs_prm->clust_size = ntfs_prm->sec_size*(int)buff[0xd];
    ntfs_prm->file_size = from16(buff+0x40, 4);

    if(ntfs_prm->file_size<128)
        ntfs_prm->file_size=ntfs_prm->clust_size*ntfs_prm-
>file_size;
    else if(ntfs_prm->file_size<256)
        ntfs_prm->file_size=pow(2, (256-ntfs_prm->file_size)%32);
    else
        ntfs_prm->file_size=0;

```

```

        if(ntfs_prm->file_size == 0 || ntfs_prm->clust_size == 0 ||
ntfs_prm->sec_size == 0)
        {
            syslog(LOG_WARNING, "Ошибка, %s:%d, чтения загрузочного
сектора %s", __FILE__, __LINE__, dev );
            free(buff);
            QMessageBox::warning( 0, "Носитель", "Ошибка чтения
загрузочного сектора");
            return NULL;
        }

        ntfs_prm->mft_addr=ntfs_prm->clust_size*from16(buff+0x30, 8);
        ntfs_prm->mft_mirror_addr=ntfs_prm-
>clust_size*from16(buff+0x38, 8);

        ::lseek( fd, ntfs_prm->mft_addr+3*ntfs_prm->file_size,
SEEK_SET );

        memset(buff, 0, boot_size);
        free(buff);
        buff = (unsigned char *)malloc(ntfs_prm->file_size);
        memset(buff, 0, ntfs_prm->file_size);

        if( ntfs_prm->file_size != (unsigned long)::read( fd, buff,
ntfs_prm->file_size) )
        {
            syslog(LOG_WARNING, "Ошибка, %s:%d, чтение %s: %m",
__FILE__, __LINE__, dev );
            free(buff);
            QMessageBox::warning( 0, "Носитель", "Ошибка чтения
накопителя");
            return NULL;
        }

```

```

::close(fd);

ntfs_prm->addr_upseq=from16(buff+0x4, 2);
ntfs_prm->size_upmass=from16(buff+0x6, 2);
ntfs_prm->attr_offset=from16(buff+0x14, 2);

// восстановление записи файла
for(int i=1;i<ntfs_prm->size_upmass;i++)
{
    for(int j=0;j<2;j++)
        buff[ntfs_prm->sec_size*i+j-2]=buff[ntfs_prm-
>addr_upseq+i*2+j];
}

// поиск атрибута метки
long type_attr=0;
while(type_attr!=0xFFFFFFFF)
{
    type_attr=from16(buff+ntfs_prm->attr_offset, 4);
    if(type_attr == 0x60)
        break;
    ntfs_prm->attr_offset+=from16(buff+ntfs_prm-
>attr_offset+4, 4);
}
if(type_attr != 0x60)
{
    syslog(LOG_WARNING, "Ошибка, %s:%d, не найдена метка на
носителе %s", __FILE__, __LINE__, dev );
    QMessageBox::warning( 0, "Носитель", "Ошибка получения
метки носителя");
    return NULL;
}

ntfs_prm->label_size = from16(buff+ntfs_prm->attr_offset+0x10,

```

```

4);

    ntfs_prm->label_offset = ntfs_prm-
>attr_offset+from16(buff+ntfs_prm->attr_offset+0x14, 2);
    ntfs_prm->label.setUtf16((ushort*) (buff+ntfs_prm-
>label_offset),ntfs_prm->label_size/2);

    memset(buff, 0, ntfs_prm->file_size);
    free(buff);

    return ntfs_prm;
}

//! функция перевода из 16-ричной последовательности в число
/**
    \param buf адрес начала последовательности
    \param size длина последовательности
    \return число
*/
unsigned long from16(unsigned char * buf, int size)
{
    if(buf == NULL || size <= 0)
        return -1;

    unsigned long value=0;

    for(int i=0; i<size; i++)
        value += pow(16,i*2)*(int)buf[i];
    return value;
}

//! функция получения информации об устройстве
/**
    \param dev файл устройства
    \param partition флаг учета разделов, partition=true -

```

возвращает информацию о SCSI устройствах с учетом разделов, иначе
- возвращает информацию о внешних устройствах без учета разделов
 \return информация об устройстве

*/

```
TStDevInfo disktest::getDevInfo(const char *dev, bool partition)
{
    TStDevInfo stDevInfo;
    stDevInfo.fs_type = NOT_SUPPORTED;
    stDevInfo.fileDev = dev;
    stDevInfo.removable = 0;
    stDevInfo.dev_type = 0;

    int fd = ::open(stDevInfo.fileDev.data(), O_RDONLY |
O_NONBLOCK);
    if(fd >= 0)
    {
        stDevInfo.devSize = ::lseek64(fd, 0, SEEK_END);

        ::close(fd);

        stDevInfo.hash = readHash(stDevInfo);

        bool ok = false;
        if(partition)
            ok = (stDevInfo.fileDev.right(1).toInt() == 0
&& !getSCSIDevInfo(&stDevInfo, partition)) ||
(stDevInfo.fileDev.right(1).toInt() != 0);

        if(ok)
        {
            int ret = getLabel(stDevInfo, partition);
            if(ret)
                stDevInfo.removable = -1;
        }
    }
}
```

```

        if(!ret&& stDevInfo.fs_type != NOT_SUPPORTED)
        {
            stDevInfo.mountPoint = stDevInfo.fileDev +
"/";

            stDevInfo.mountPoint.replace("dev",
"mnt");

            QDir dir(stDevInfo.mountPoint.data());
            if(dir.exists())

dir.rmpath(stDevInfo.mountPoint.data());

        }
    }
    else
        stDevInfo.removable = -1;
}
else if(partition)
{
    syslog(LOG_WARNING, "Ошибка, %s:%d,
устройство \"%s\": %m", __FILE__, __LINE__,
stDevInfo.fileDev.data());

    QMessageBox::warning( 0, "Носитель", "Ошибка
чтения накопителя");

    stDevInfo.removable = -1;
}
return stDevInfo;
}

//! функция заполнения списка устройств
/**
    \param dev_info_list список устройств
    \return результат выполнения
*/
QList<TStDevInfo> disktest::mkPartitionList(QList<TStDevInfo>&
dev_info_list)

```

```

{
    QList<TStDevInfo> dev_list;
    QString m_dev = "", t_str = "";
    // Пробежимся по всем файлам устройств для того, чтобы в
системе определились флешки
    char ud_sd[]="/dev/sda";
    char rb, rb_len;
    for(ud_sd[7]='a'; ud_sd[7]!='z'; ud_sd[7]++)
    {
        int ffd = ::open(ud_sd, O_RDONLY, 0444);
        if( ffd != -1 )
        {
            rb_len=::read(ffd, &rb, 1);
            ::close(ffd);
        }
    }
    /// Формирование списка устройств
    QFile f_dev("/proc/partitions");
    if(f_dev.open(QIODevice::ReadOnly))
    {
        QTextStream tStream(&f_dev);

        // пропускаем первые две строчки
        m_dev = tStream.readLine();
        m_dev = tStream.readLine();

        while(!tStream.atEnd())
        {
            m_dev = tStream.readLine();

            t_str = m_dev.section(' ', -1);

            if(
                t_str.contains("loop")
                || t_str.contains("sr")
            )

```

```

        || t_str.contains("etherd")
        || t_str.contains("mtdblock")
        || t_str.contains("mmcblk0")
        || t_str.contains("ram")
    )
    continue;

    if( !
QFile::exists( QByteArray("/dev/").append(t_str).data() ) )
    {
        unsigned int major = m_dev.section( ' ', 0, 0,
QString::SectionSkipEmpty ).toInt();
        unsigned int minor = m_dev.section( ' ', 1, 1,
QString::SectionSkipEmpty ).toInt();

        if( mknod(QByteArray("/dev/").append(t_str).data(), 0600 |
S_IFBLK, makedev(major, minor)) )
        {
            syslog(LOG_ERR, "Ошибка, %s:%d, Не удалось
создать устройство %s: %m", __FILE__, __LINE__,
QByteArray("/dev/").append(t_str).data());
            QMessageBox::information(0, "", "Не удалось
создать устройство " + QByteArray("/dev/").append(t_str));
        }
    }

    TStDevInfo stDevInfo =
getDevInfo(QByteArray("/dev/").append(t_str).data(), true);
    if( stDevInfo.removable != -1 ) {
        dev_list += stDevInfo;
    } else {
        syslog(LOG_INFO, "%s removable == -1",
t_str.data());
    }
}

```

```

        }

        f_dev.close();
    }
    else
        QMessageBox::information(0, "", "Не удалось открыть файл "
+ f_dev.fileName());

    dev_info_list = dev_list;

    return dev_list;
}

//! функция получения информации о SCSI-устройстве
/**
    \param fd дескриптор устройства
    \param dev_info структура параметров устройства
    \param partition флаг учета разделов, partition=true -
извлекает информацию с учетом разделов, иначе - извлекает
информацию без учета разделов
    \return результат выполнения
*/
int disktest::getSCSIDevInfo(TStDevInfo *dev_info, bool
partition )
{
    struct sg_io_hdr io_hdr;
    unsigned char sense_b[32];
    unsigned char buff[128];
    unsigned char CmdBlk[] = {0, 0, 0, 0, 0, 0};
    unsigned char dev_type;
    int fd_p, fd;

    memset(&io_hdr, 0, sizeof( struct sg_io_hdr ));

```

```

memset(&buff, 0, 128);
io_hdr.interface_id = 'S';
io_hdr.dxfer_direction = SG_DXFER_FROM_DEV;
io_hdr.dxfer_len = 128;
CmdBlk[0] = INQUIRY;
CmdBlk[4] = 128;
io_hdr.cmd_len = sizeof( CmdBlk );
io_hdr.mx_sb_len = sizeof( sense_b );
io_hdr.dxferp = buff;
io_hdr.cmdp = CmdBlk;
io_hdr.sbp = sense_b;
io_hdr.timeout = 5000;

if( -1 == (fd = ::open( dev_info->fileDev.data(), O_RDONLY |
O_NONBLOCK )) )
{
    syslog(LOG_WARNING, "Ошибка, %s:%d, %s %m", __FILE__,
__LINE__, dev_info->fileDev.data());
    return -1;
}

//посылаем команду INQUIRY (описание в "SCSI primary commands-
4")
if( !ioctl( fd, SG_IO, &io_hdr ) )
{
    if( ( ( buff[0] & 0xE0 ) == 0x20 ) || ( ( buff[0] & 0xE0 )
== 0x60 ) )
    {
        if(partition)
        {
            syslog(LOG_WARNING, "Ошибка, %s:%d, устройство не
подключено или не поддерживается", __FILE__, __LINE__);
            QMessageBox::warning( 0, "Носитель", "Ошибка
чтения накопителя");

```

```

    }

    ::close(fd);
    return -1; //устройство не подключено или не
поддерживается
}

if( !( buff[1] & 0x80) )
{
    dev_info->removable = 0;
}
else
    dev_info->removable = 1;

// определяем тип устройства
dev_type = buff[0] & 0x1F;

if( ( dev_type != 0x00 ) && ( dev_type != 0x0E ) )
{
    if(partition)
    {
        syslog(LOG_WARNING, "Ошибка, %s:%d, устройство не
прямого доступа", __FILE__, __LINE__);
        QMessageBox::warning( 0, "Носитель", "Ошибка
чтения накопителя");
    }
    ::close(fd);
    return -1; //работа только с устройствами прямого
доступа
}

//определяем изготовителя
dev_info->vendor = QByteArray((const char*)&buff[8],
8).trimmed();

```

```

        if(dev_info->vendor.isEmpty())
            dev_info->vendor = "USB 2.0";

// sata устройства определяются как removable
if((dev_info->vendor == "ATA") || (dev_info->vendor == "ata"))
    dev_info->removable = 0;

//определяем модель устройства
dev_info->model = QByteArray((const char*)&buff[16],
16).trimmed();
if(dev_info->removable == 1)
    dev_info->dev_type = VNI_FLASH;
else
    dev_info->dev_type = VNI_HARDDISK;

getStorageUSBSerial(dev_info);
}
else
{
    if(partition)
    {
        int err = errno;
        if(err != ENOTTY) {
            syslog(LOG_WARNING, "Ошибка, %s:%d, %m (%d) %s",
__FILE__, __LINE__, err, dev_info->fileDev.data());
            QMessageBox::warning( 0, "Носитель", "Ошибка
чтения накопителя");
        }
    }
    ::close(fd);
    return -1;
}
if(!partition)

```

```

{

    //определяем, есть ли на устройстве раздел
    fd_p = ::open( QByteArray(dev_info-
>fileDev).append("1").data(), O_RDONLY | O_NONBLOCK );
    if( fd_p >= 0 )
    {
        dev_info->fileDev.append('1');
        ::close( fd_p );
    }

    fd_p = ::open( QByteArray(dev_info-
>fileDev).append("2").data(), O_RDONLY | O_NONBLOCK );
    if( fd_p >= 0 )
    {
        dev_info->fileDev.append('2');
        ::close( fd_p );
    }
}

//определяем, готово ли устройство
io_hdr.dxfer_direction = SG_DXFER_NONE;
io_hdr.dxfer_len = 0;
CmdBlk[0] = TEST_UNIT_READY;
CmdBlk[4] = 0;

if( !ioctl( fd, SG_IO, &io_hdr ) )
    if( io_hdr.sb_len_wr )
        if( ( ( sense_b[0] & 0x70 ) == 0x70 ) &&
( ( sense_b[2] & 0x0F ) == 2 ) )
        {
            ::close(fd);
            return -1;    //нет носителя
        }
::close(fd);

```

```

        return 0;
    }

int disktest::getStorageUSBSerial(TStDevInfo *dev_info)
{
    if(dev_info->dev_type == VNI_HARDDISK)
        return 0;

    QByteArray nDev;
    int i = dev_info->fileDev.size();
    while (dev_info->fileDev[i] != '/')
    {
        nDev.insert(0,dev_info->fileDev[i--]);
    }

    QString str = "/sys/block/" + nDev;
    char *f = (char *)malloc(255);
    memset(f, 0, 255);
    readlink(str.toAscii().data(), f, 255);

    QString strHostNum;
    QString Num;

    strHostNum.append((const char*)f);
    free(f);

    int y = strHostNum.indexOf("/host")+5;

    while (strHostNum[y] != '/')
        Num.append(strHostNum[y++]);

    int fd_sys = 0;
    QString path = QString("/proc/scsi/usb-storage/") + Num;
    fd_sys = ::open((const char*)path.toAscii().data(), O_RDONLY |

```

```

O_NONBLOCK);
    if(fd_sys == -1)
    {
        syslog(LOG_WARNING, "Ошибка, %s:%d, %s %m", __FILE__,
__LINE__, path.toAscii().data());
        return -1;
    }

    const unsigned int fsize = 512;
    char *buf = (char *)malloc(fsize);
    ::read(fd_sys, buf, fsize);
    QString SerialNumber;
    QStringList StrList = QString(buf).split("\n",
QString::SkipEmptyParts);

    ::close(fd_sys);
    free(buf);

    for (int p = 0; p < StrList.count(); p++)
    {
        QStringList ParamList = StrList[p].split(":");
        if (ParamList.count() > 0 && ParamList.count() <= 2)
        {
            if (ParamList[0].trimmed().toLowerCase() ==
QString("Serial Number").toLowerCase())
            {
                SerialNumber=ParamList[1].trimmed();
                break;
            }
        }
        else
            continue;
    }

```

```

        dev_info->serialNumber = QByteArray((const
char*)SerialNumber.toAscii().data(),
SerialNumber.toAscii().size());

        return 0;
}

```

Код файла giz.cc:

```

//функция вычисления Контрольной Суммы (ГОСТ)

#include <string.h>
#include <stdio.h>
#include <syslog.h>

#define giz(...) CAT(giz_, COUNT_PARMS(__VA_ARGS__))(__VA_ARGS__)
#define giz_2(A, B) giz_3(A, B, NULL)

#define CAT(A, B) CAT2(A, B)
#define CAT2(A, B) A ## B

#define COUNT_PARMS2(_1, _2, _3, _, ...) _
#define COUNT_PARMS(...) COUNT_PARMS2(__VA_ARGS__, 3, 2, 1)

static unsigned int kzu[8] =
{ 0x671db88c, 0x7d79ab1f, 0x43de5890, 0x29b16c2e,      // K1
  0x009fa589, 0x627644c9, 0x479a9460, 0x554bce2b      };

unsigned char zamena[16][8]=
{
    0x09, 0x06, 0x03, 0x02, 0x08, 0x0b, 0x01, 0x07,
    0x0a, 0x04, 0x0e, 0x0f, 0x0c, 0x00, 0x0d, 0x05,

    0x03, 0x07, 0x0e, 0x09, 0x08, 0x0a, 0x0f, 0x00,
    0x05, 0x02, 0x06, 0x0c, 0x0b, 0x04, 0x0d, 0x01,

```

```

0x0e, 0x04, 0x06, 0x02, 0x0b, 0x03, 0x0d, 0x08,
0x0c, 0x0f, 0x05, 0x0a, 0x00, 0x07, 0x01, 0x09,

0x0e, 0x07, 0x0a, 0x0c, 0x0d, 0x01, 0x03, 0x09,
0x00, 0x02, 0x0b, 0x04, 0x0f, 0x08, 0x05, 0x06,

0x0b, 0x05, 0x01, 0x09, 0x08, 0x0d, 0x0f, 0x00,
0x0e, 0x04, 0x02, 0x03, 0x0c, 0x07, 0x0a, 0x06,

0x03, 0x0a, 0x0d, 0x0c, 0x01, 0x02, 0x00, 0x0b,
0x07, 0x05, 0x09, 0x04, 0x08, 0x0f, 0x0e, 0x06,

0x01, 0x0d, 0x02, 0x09, 0x07, 0x0a, 0x06, 0x00,
0x08, 0x0c, 0x04, 0x05, 0x0f, 0x03, 0x0b, 0x0e,

0x0b, 0x0a, 0x0f, 0x05, 0x00, 0x0c, 0x0e, 0x08,
0x06, 0x02, 0x03, 0x09, 0x01, 0x07, 0x0d, 0x04
};

//! вычисление имитозащиты по ГОСТ 28147-89
/*! \param *text буфер данных
    \param size размер данных
    \param *mask маска, накрывающая данные
    \return вычисленная контрольная сумма
*/
unsigned int giz_3( unsigned char* text, int size, unsigned char*
mask )
{

    unsigned int N1, N2;
    unsigned int CM1, CM2, R;

    int i, k;
    unsigned int *ot1;

```

```

unsigned int *m;

unsigned int tail[2] = {0,}; // массив из 2 элементов (второй
пустой)
unsigned int tail_mask[2] = {0,};

ot1 = (unsigned int*)text; // записывается буфер данных
m = (unsigned int*)mask; // записывается маска накрывающая
данные

if( size%8 ) //если размер данных текста не кратен 8
{
    memcpy( (char*)tail, text+((size/8)*8), size -
( (size/8)*8) );
    if(mask)
        memcpy( (char*)tail_mask, mask, size -
( (size/8)*8) );
}

k = 0;
N1 = 0;
N2 = 0;

while(size > 0)
{
    if( size > 7 )
    {
        //обработка основного массива текста (полными
восмерками)
        N1 = mask?N1^ot1[k]^m[0]:N1^ot1[k]; //если маска? то
N1 xor Буфер данных xor маска, иначе без маски (снимаем маску,
если она есть)
        N2 = mask?N2^ot1[k+1]^m[1]:N2^ot1[k+1];
    }
}

```

```

else
{
    //обработка хвоста текста (неполной восьмерки)
    N1 = mask?N1^tail[0]^tail_mask[0]:N1^tail[0];
    N2 = mask?N2^tail[1]^tail_mask[1]:N2^tail[1];
}

{ // zikl16();
    int i_16, j_16;
    for(i_16 = 0; i_16 < 2; i_16++)
    {
        for(j_16 = 0; j_16 < 8; j_16++)
        {
            { // zikl1(j);
                int m,n,l;

                CM1 = N1 + kzu[j_16];    // сложение по mod
2^32

                R = 0L;

                for(m = 7; m >= 0; m--)
                {
                    n = 4*m; // m от 7 до 0
                    l = (CM1>>n)&0xf; // сдвиг вправо на n
бит и запись в переменную последних 4ех бит

                    R = R<<4; // сдвиг вправо на 4 бита
                    R = R^zamena[l][7-m]; // r xor
значение таблицы(л столбец, 7-м строка)
                }
                m = R>>21; // сдвиг на 21 разр. вправо и
запись

                R = (R<<11) + m;
                CM2 = R^N2; // побитно по мод 2
            } // zikl1(j);

```

```

        N2 = N1;
        N1 = CM2;
    }
}
} // zikl16();
k += 2;
size -= 8;
}
return (N1);
}

//! вычисление свертки пароля
/*! \param *text буфер пароля (serialNumber)
    \param *keyset ключ свертки (hash)
    \return свертка пароля
*/
char* passwordhash(char* text, char* keyset)
{
    unsigned int ret;
    static char rezult[13];
    char* salt = keyset;
    char* kzuinit = (char*)kzu;
    if(!strlen(text) || !strlen(keyset))
        return NULL;
    if(keyset[0] == '$')
        salt++;
    kzuinit[0] = salt[0];
    kzuinit[1] = salt[1];
    ret = giz((unsigned char*)text, strlen(text), 0); //контрольная
сумма
    sprintf(rezult, "$%.2s$%X\0", salt, ret); // $, преобразование
2ух символов из salt, $, 16рич знач ret
    return rezult;}

```

Приложение Д

(обязательное)

Код проекта, отвечающего за форматирование устройства

Код заголовочного файла fdrive.h:

```
//! \file fdrive.h Файл содержит объявление класса главного окна приложения fdrive.
```

```
#ifndef FDRIVE_H
```

```
#define FDRIVE_H
```

```
#include <QDialog>
```

```
#include <QProcess>
```

```
#include <QEvent>
```

```
namespace Ui {
```

```
    class FDrive;
```

```
}
```

```
class FDrive : public QDialog {
```

```
    Q_OBJECT
```

```
public:
```

```
    FDrive(QWidget *parent = 0);
```

```
    ~FDrive();
```

```
    int init(QString dev, bool vnio = false);
```

```
    void fEnabled(bool);
```

```
protected:
```

```
    void changeEvent(QEvent *e);
```

```
private:
```

```
    QString _dev;
```

```
    QString m_format;
```

```

QStringList m_arg;
QProcess *m_proc;
int UEvent;
int max_size;
bool m_vnio;

Ui::FDrive *ui;
enum {UNKNOWN = 0, FAT12, FAT16, FAT32};

private slots:
    void on_b_close_clicked();
    void on_b_format_clicked();

    void procFinished(int, QProcess::ExitStatus);
    void procError(QProcess::ProcessError);
    void procRRStdErr();
    void procRRStdOut();

protected slots:
    void closeEvent(QCloseEvent*);
};

#endif // FDRIVE_H

    Код файла main.cpp:
#include <QtGui/QApplication>
#include <QVariant>
#include "fdrive.h"

#include <QTextCodec>

#include <syslog.h>
#define LOGERRm      LOG_ERR, "Ошибка, %s:%d, %m", __FILE__,
__LINE__
#define LOGERRs      LOG_ERR, "Ошибка, %s:%d, %s", __FILE__,

```

__LINE__

//! главная функция приложения.

/**

Главная функция приложения. Создает объекты приложения и инициализирует его.

\param argc количество аргументов.

\param argv указатель на аргументы.

\return код возврата.

*/

int main(int argc, char *argv[])

{

QTextCodec::setCodecForCStrings(QTextCodec::codecForName("UTF-8"));

openlog("fdrive", LOG_PID, LOG_USER);

QApplication a(argc, argv, QApplication::GuiClient);

FDrive w;

bool vnio = false;

if(argc > 2)

{

QString param(argv[2]);

if(param.toInt() > 0)

vnio = true;

}

if(w.init(QVariant(argv[1]).toString(), vnio))

{

w.show();

return a.exec();

}

}

Код файла fdrive.cpp:

```
//! \file fdrive.cpp реализация функций главного окна приложения.

#include "fdrive.h"
#include "ui_fdrive.h"

#include <QMessageBox>
#include <QtDebug>
#include <QTextCodec>
#include <QFile>
#include <QDir>

#include <sys/mount.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include "fstream"

#include <syslog.h>

#define BUFFSIZE      512
#define aName          "Форматирование накопителя"
#define ICONDIR        "../share/shell/"
#define ICONFIL        "disktest.png"

using namespace std;

//! Конструктор класса FDrive.
/**
 * Создает объекты элементов управления и инициализирует их.
 * \param parent указать на родительское окно.
 */
FDrive::FDrive(QWidget *parent) :
```

```

        QDialog(parent),
        m_vnio(false),
        ui(new Ui::FDrive)
    {
        ui->setupUi(this);

        setWindowTitle(aName);

setWindowIcon(QIcon(QString("%1%2").arg(ICONDIR).arg(ICONFIL)) );

        _dev = "";
        m_format = "/home/vladimir/project1/mkdosfs/mkdosfs";

        Qt::WindowFlags flags;
        flags |= Qt::Dialog;
        flags |= Qt::WindowTitleHint;
        flags |= Qt::WindowCloseButtonHint;
        setWindowFlags(flags);
        setFixedSize(width(), height());
    }

    //! Деструктор класса FDrive.
    FDrive::~FDrive()
    {
        closelog();

        delete ui;
    }

    //! Переводит имена элементов приложения.
    /**
    Используется для многоязыковой поддержки.
    */

```

```

void FDrive::changeEvent(QEvent *e)
{
    QDialog::changeEvent(e);
    switch (e->type()) {
    case QEvent::LanguageChange:
        ui->retranslateUi(this);
        break;
    default:
        break;
    }
}

//! Инициализация основного окна приложения.
/**
    \param dev название устройства.
 */
int FDrive::init(QString dev, bool vnio)
{
    QByteArray m_label;
    _dev = dev;
    m_arg.clear();
    m_arg << "-I" << "-F" << "32";

    int fd, nResult, fs = UNKNOWN;
    char buff[BUFFSIZE];

    ui->progressBar->setMinimum(0);

    do
    {
        fd = ::open( dev.toAscii(), O_RDONLY );
        nResult = 1;
        if( -1 == fd )
            nResult = QMessageBox::warning ( this, aName, "He mory

```

```

открыть накопитель для чтения", "&Повторить", "Отмена" );
    }
    while( !nResult );

    if( -1 == fd )
    {
        ::close(fd);
        return 0;
    }

    // Определяем размер девайса
    off64_t res_size = ::lseek64(fd, 0, SEEK_END);
    max_size = res_size >> 10;
    qWarning ("max_size - %d", max_size);
    ui->progressBar -> setMaximum(max_size);

    ::lseek( fd, 0, SEEK_SET );
    //Читаем mbr для проверки файловой системы
    if( BUFFSIZE != ( nResult = ::read( fd, buff, BUFFSIZE) ) )
    {
        QMessageBox::warning( this, aName, "Не могу прочитать
mbr", QMessageBox::Ok, QMessageBox::NoButton );
        ::close(fd);
        return 0;
    }

    //Проверяем файловую систему FAT12
    if( 'F' != buff[54] || 'A' != buff[55] || 'T' != buff[56] ||
'1' != buff[57] || '2' != buff[58] )
    {
        //Проверяем файловую систему FAT16
        if( 'F' != buff[54] || 'A' != buff[55] || 'T' != buff[56]
|| '1' != buff[57] || '6' != buff[58] )
        {

```

```

        //Проверяем файловую систему FAT32
        if( 'F' != buff[82] || 'A' != buff[83] || 'T' !=
buff[84] || '3' != buff[85] || '2' != buff[86] )
            fs = UNKNOWN;
        else
            fs = FAT32;
    }
    else
        fs = FAT16;
}
else
    fs= FAT12;

if( FAT32 == fs )
{
    buff[82] = '\0';
    m_label = QByteArray(buff + 71, 11);
}
else
{
    buff[54] = '\0';
    m_label = QByteArray(buff + 43, 11);
}

int zero_offset = 0;
if( m_label[0] == '\0' )
    zero_offset = 1;

m_label = QTextCodec::codecForName("KOI8-R")->
toUnicode(m_label.mid(zero_offset)).toAscii();

// vnio - внешний носитель информации открытый
if(vnio)
{

```

```

        QRegExp regexp("Flash\\d{0,}");
        QRegExpValidator *validator = new QRegExpValidator(regexp,
ui->lineEdit());
        m_vnio = true;

        ui->lineEdit->setValidator(validator);
        int pos = 0;
        QString label = QString(m_label);

        if(validator->validate(label, pos) ==
QValidator::Acceptable)
            ui->lineEdit->setText(m_label);
        else
            ui->lineEdit->setText("Flash");
    }
else
{
    if(!m_label.isEmpty())
        ui->lineEdit->setText(m_label);
    else
        ui->lineEdit->setText("NO NAME");
}

    return 1;
}

//! Обрабатывает нажатие кнопки Форматировать.
/**
*/
void FDrive::on_b_format_clicked()
{
    if(m_vnio)
    {
        QRegExp regexp("Flash\\d{0,}");

```

```

    QRegExpValidator validator(regexp, 0);
    int pos = 0;
    QString label = ui->lineEdit->text();

    if ((label.length() <= 0) ||
        (validator.validate(label, pos) !=
QValidator::Acceptable))
    {
        QMessageBox::critical(this, "Ошибка", "Введена
неправильная метка");
        reject();
        return ;
    }
}

setEnabled(FALSE);

if (ui->lineEdit->text().length() > 11)
{
    QMessageBox::warning(this, aName,    // "Формат Диск " +
cmbBox1->currentText(),
                        "Указана недопустимая метка тома.
Введите правильную метку тома.", "&Заккрыть");
    enabled(TRUE);
    ui->lineEdit->selectAll();
    ui->lineEdit->setFocus();
}
else
{
    if (!_dev.isEmpty() && !_dev.isNull())
    {
        m_proc = new QProcess(this);
        if (m_proc)
        {

```

```

        ui->progressBar->setValue(max_size / 100); //
первый шаг прогресс бара на 1 процент
        update();
        QApplication->processEvents();

        if(ui->checkBox->checkState() == Qt::Unchecked)
            m_arg << "-c";

        m_arg << QString("-n");
        m_arg << ui->lineEdit->text();
        m_arg << _dev;

        connect(m_proc,
        SIGNAL(finished(int,QProcess::ExitStatus)), this,
        SLOT(procFinished(int,QProcess::ExitStatus)));
        connect(m_proc,
        SIGNAL(error(QProcess::ProcessError)), this,
        SLOT(procError(QProcess::ProcessError)));
        connect(m_proc, SIGNAL(readyReadStandardError()),
        this, SLOT(procRRStdErr()));
        connect(m_proc, SIGNAL(readyReadStandardOutput()),
        this, SLOT(procRRStdOut()));

        QTextCodec::setCodecForLocale(QTextCodec::codecForName("KOI8-R"));
        m_proc->start(m_format, m_arg); // вызов mkdosfs
        QTextCodec::setCodecForLocale(0); // to default

        if(!m_proc->waitForStarted())
            qWarning("proc not started");
        if(!m_proc->waitForReadyRead(-1))
            qWarning("can't read stdout");
        if(!m_proc->waitForFinished(-1))
        {
            if(m_proc->state() == QProcess::NotRunning)

```



```

}

//! Обработка завершения процесса.
/**
    \param exit_code код завершеного процесса.
    \param exit_status статус завершеного процесса.
*/
void FDrive::procFinished(int exit_code, QProcess::ExitStatus
exit_status)
{
    disconnect(m_proc, SIGNAL(finished(int,QProcess::ExitStatus)),
this, SLOT(procFinished(int,QProcess::ExitStatus)));
    disconnect(m_proc, SIGNAL(error(QProcess::ProcessError)),
this, SLOT(procError(QProcess::ProcessError)));
    disconnect(m_proc, SIGNAL(readyReadStandardError()), this,
SLOT(procRRStdErr()));
    disconnect(m_proc, SIGNAL(readyReadStandardOutput()), this,
SLOT(procRRStdOut()));

    switch (exit_status)
    {
    case QProcess::CrashExit:
    {
        QMessageBox::critical(this,aName,QString("Ошибка запуска
приложения mkdosfs"));
        //syslog?
        fEnabled(TRUE);

        if(m_proc)
            delete m_proc;

        m_proc = 0;

        init(_dev);
    }
    }
}

```

```

        reject();
        break;
    }
    case QProcess::NormalExit:
    {
        if(exit_code != 0)
        {
            QMessageBox::critical(this, aName, QString("Ошибка
запуска приложения mkdosfs"));
            //syslog?
            fEnabled(TRUE);

            if(m_proc)
                delete m_proc;

            m_proc = 0;
            init(_dev);
            reject();
        }
        else
        {
            qWarning("Normal exit");
            fEnabled(TRUE);

            if(m_proc)
                delete m_proc;

            m_proc = 0;

            QMessageBox::information(this, aName, "Форматирование
завершено");

            ui->progressBar->setValue(0);

```

```

        init(_dev);

        accept();
    }
    break;
}
}

}

//! Обработка ошибок процесса.
/**
    \param proc_err тип ошибки.
*/
void FDrive::procError(QProcess::ProcessError proc_err)
{
    qWarning("procerror ----- ");
    fEnabled(TRUE);

    switch(proc_err)
    {
    case QProcess::FailedToStart:
    {
        qWarning("mkdosfs not found");
        break;
    }
    case QProcess::Crashed:
    {
        qWarning("Crashed");
        break;
    }
    case QProcess::Timedout:
    {
        qWarning("Time is out");
        break;
    }
    }
}

```

```

    }
    case QProcess::WriteError:
    {
        qWarning("Write error");
        break;
    }
    case QProcess::ReadError:
    {
        qWarning("Read error");
        break;
    }
    case QProcess::UnknownError:

    {
        qWarning("Unknown error");
        break;
    }
}

//! Обработка стандартного потока ошибок процесса.
/**
 */
void FDrive::procRRStdErr()
{
    qWarning("procRRStdErr");
    QByteArray out = m_proc->readAllStandardError();
    QString strout = QString::fromLocal8Bit(out);
    qWarning() << strout;
}

```

```

//! Обработка стандартного потока ввода\вывода процесса.
/**
*/
void FDrive::procRRStdOut()
{
    QByteArray out = m_proc->readAllStandardOutput();

    bool ok;
    int m_pb = out.toInt(&ok,10);

    if(ok)
    {
        if(m_pb >= ui->progressBar->maximum())
            ui->progressBar->setValue(ui->progressBar->maximum());
        else
        {
            ui->progressBar->setValue(m_pb);
            ui->progressBar->update();
            //      QApplication->processEvents();
        }
    }
}

//! обработка события закрытия диалога
/**
*/
void FDrive::closeEvent(QCloseEvent *)
{
    accept();
}

```

Приложение Е

(обязательное)

Протокол проверки на оригинальность



АНТИПЛАГИАТ
ТВОРИТЕ СОБСТВЕННЫМ УМОМ

Пензенский государственный
университет

СПРАВКА

о результатах проверки текстового документа
на наличие заимствований

Проверка выполнена в системе
Антиплагиат.ВУЗ

Автор работы	Мещеряков Владимир Алексеевич
Подразделение	Кафедра ИБСТ ПГУ
Тип работы	Выпускная квалификационная работа
Название работы	ЗАЩИТА ОТ НЕСАНКЦИОНИРОВАННОГО ДОСТУПА ИЗДЕЛИЙ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ СПЕЦИАЛЬНОГО НАЗНАЧЕНИЯ С ПРИМЕНЕНИЕМ ДОВЕРЕННЫХ ОТЧУЖДАЕМЫХ НОСИТЕЛЕЙ ИНФОРМАЦИИ
Название файла	15ПИ1_МещеряковВА_ПЗ.pdf
Процент заимствования	15.31 %
Процент самоцитирования	0.00 %
Процент цитирования	3.45 %
Процент оригинальности	81.24 %
Дата проверки	14:46:33 07 июня 2020г.
Модули поиска	Модуль поиска ИПС "Адилет"; Модуль выделения библиографических записей; Сводная коллекция ЭБС; Модуль поиска "Интернет Плюс"; Коллекция РГБ; Цитирование; Модуль поиска переводных заимствований; Модуль поиска переводных заимствований по eLibrary (EnRu); Модуль поиска переводных заимствований по интернет (EnRu); Коллекция eLIBRARY.RU; Коллекция ГАРАНТ; Коллекция Медицина; Диссертации и авторефераты НББ; Модуль поиска перефразирований eLIBRARY.RU; Модуль поиска перефразирований Интернет; Коллекция Патенты; Модуль поиска "ПГУ"; Модуль поиска общеупотребительных выражений; Кольцо вузов
Работу проверил	Иванов Алексей Петрович ФИО проверяющего
Дата подписи	 Подпись проверяющего

Чтобы убедиться
в подлинности справки,
используйте QR-код, который
содержит ссылку на отчет.



Ответ на вопрос, является ли обнаруженное заимствование
корректным, система оставляет на усмотрение проверяющего.
Предоставленная информация не подлежит использованию
в коммерческих целях.



Руководитель ВКР
«8» 06 2020 г.

Подпись Баркунов А.В. удостоверяю
Начальник бюро кадров Т.Т. Стальнова

Баркунов А.В.

Рисунок Е.1 — Протокол проверки на оригинальность