Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина «Архитектура вычислительных систем»

«К ЗАЩИТЕ ДОПУСТИТЬ»
Руководитель курсового проекта
магистр техн.наук, ассистент
_____А.В. Давыдчик
___._____.2023

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту
на тему:

«ПЕРЕХВАТ И ПРОТОКОЛИРОВАНИЕ СЕТЕВОГО ТРАФИКА»

БГУИР КП 1-40 01 01 034 ПЗ

Выполнил студент группы 053504
Матвеев Илья Андреевич

_____
(подпись студента)
Курсовой проект представлен на проверку ___._____.2023

_____
(подпись студента)

Минск 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современный мир невозможно представить без сетевых технологий. Интернет, сетевые приложения и сервисы используются повсеместно в различных сферах деятельности, начиная от личной жизни и заканчивая крупными корпоративными сетями. Однако, с ростом количества передаваемой информации и разнообразия способов ее передачи, возрастает необходимость в контроле и обеспечении безопасности передаваемых данных.

В рамках данной работы было рассмотрено одно из наиболее эффективных средств контроля - перехват и протоколирование сетевого трафика. Цель данного курсового проекта заключается в разработке программного продукта, осуществляющего перехват и протоколирование сетевого трафика в компьютерной сети.

В работе будут рассмотрены теоретические аспекты перехвата и протоколирования сетевого трафика, обзор и сравнительный анализ существующих программных продуктов, а также разработка и реализация собственной программы с описанием ее архитектуры и функциональных возможностей.

Результаты данного исследования помогут в обеспечении безопасности и контроле сетевого трафика, а также будут полезны в обучении студентов и специалистов в области информационной безопасности и сетевых технологий.

1 АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ

1.1 Структура и архитектура вычислительной системы

Базовая структура вычислительной системы включает следующие блоки:

– Блок процессора;

– Блок памяти, который состоит из оперативной и постоянной части;

– Блок ввода-вывода данных, служащий для информационного обмена с внешними устройствами.

Все блоки вычислительной системы объединяет общая шина или по-другому информационный канал, или системная магистраль. На рисунке 1 изображена обобщённая архитектура вычислительной системы.

Рисунок 1 – Архитектура вычислительной системы

В состав системной магистрали входят четыре шины, которые являются шинами низкого уровня:

– Шина адреса;

– Информационная шина или шина данных;

– Шина управления;

– Шина питания.

Основные понятия, связанные со структурной организацией вычислительных систем, следующие:

– Электронной системой называют любое электронное устройство, которое предназначено для работы с информацией;

– Задачей является список действий, подлежащих исполнению при помощи электронных систем;

– Быстродействие – это параметры скорости осуществления внутри электронной системой возложенных на неё функций;

– Гибкость системы – это свойство системы перенастраиваться для осуществления различных задач;

– Избыточность системы – это соответствие уровня сложности задач, подлежащих решению, технологическим параметрам системы;

– Системный интерфейс – это набор условий информационного обмена, который подразумевает электронную, а также на основе структуры и логики, способность обмениваться данными между разными модулями, способными участвовать в этом процессе.

Вычислительной или микропроцессорной системой является разновидность электронных систем, которые предназначены для анализа входной информации и пересылки выходных данных. На рисунке 2 изображена блок-схема вычислительной системы.

Рисунок 2 – Блок-схема вычислительной системы

1.2 История, версии и достоинства

Intel Coffee Lake (8 поколение) – кодовое название семейства процессоров Intel Core восьмого поколения.

Согласно принятой в компании стратегии разработки микропроцессоров «Тик-так», вслед за «тиком» Broadwell последовал «так» в виде Skylake, Kaby Lake и Coffee Lake с незначительным изменением техпроцесса с 14-нм до 14-нм+.

Чипы официально анонсированы 24 сентября 2017 года и стали доступны для покупки, начиная с 5 октября 2017 года.

Основным отличием семейства процессоров от предыдущего поколения является увеличение до шести количества ядер в настольных (Coffee Lake-S) и мобильных (Coffee Lake-H) вариантах процессора.

Тепловой пакет (TDP) для настольных процессоров составляет 95 Вт, мобильных - до 45 Вт, а «ультрабучной» категории Coffee Lake-U - 28 Вт. Настольные процессоры имеют встроенную графику Intel UHD Graphics 630 с eDRAM и поддержкой DP 1.2 на HDMI 2.0 и HDCP 2.2 [2].

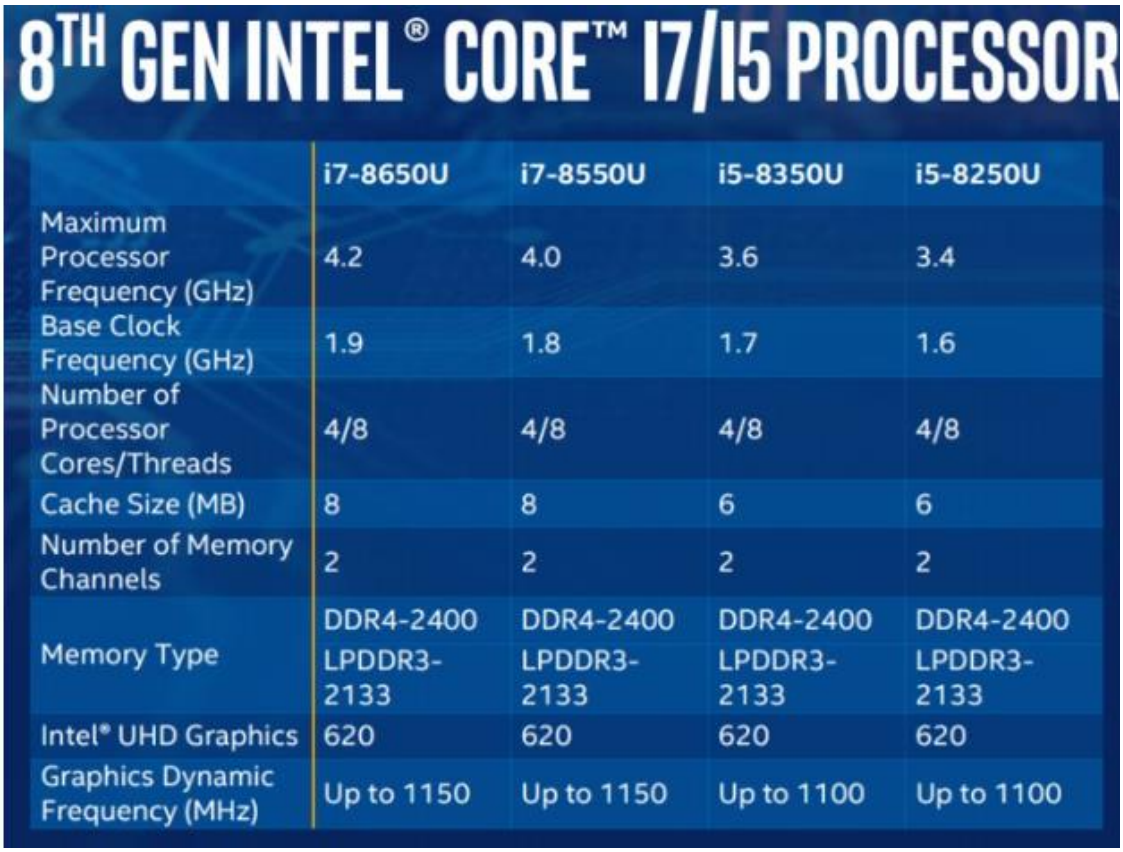
Производительность процессоров Coffee Lake на 15 % больше, по сравнению с процессорами Kaby Lake. Компания Intel сообщает о 30-процентном приросте производительности процессоров Coffee Lake в тесте SYSmark 2014, в сравнении с процессорами Skylake U-серии с TDP 15 Вт.

Процессоры Core восьмого поколения работают на относительно низких основных тактовых частотах (не выше 1,9 ГГц у старшей модели i7-8650U), благодаря чему все модели укладываются в термопакет (TDP) до 15 Вт при четырех вычислительных ядрах.

В то же время, благодаря технологии Intel Turbo Boost Technology 2.0, чипы способны динамически наращивать тактовую частоту более чем в два раза (до 4,2 ГГц у старшей модели i7-8650U), что позволяет значительным образом увеличивать производительность системы по необходимости и оставаться в «холодном» состоянии в режиме ожидания.

Благодаря двум дополнительным ядрам, производительность может быть до 50% выше, по сравнению с процессором семейства Kaby Lake на аналогичной частоте (Core i7-7820HQ, 2.9-3.9 ГГц) [4]. Одноядерная производительность осталась на уровне предшественника из семейства Kaby Lake.

На рисунке 3 изображена сравнительная характеристика первых четырех процессоров Core 8 поколения.



8TH GEN INTEL® CORE™ I7/I5 PROCESSOR

	i7-8650U	i7-8550U	i5-8350U	i5-8250U
Maximum Processor Frequency (GHz)	4.2	4.0	3.6	3.4
Base Clock Frequency (GHz)	1.9	1.8	1.7	1.6
Number of Processor Cores/Threads	4/8	4/8	4/8	4/8
Cache Size (MB)	8	8	6	6
Number of Memory Channels	2	2	2	2
Memory Type	DDR4-2400 LPDDR3-2133	DDR4-2400 LPDDR3-2133	DDR4-2400 LPDDR3-2133	DDR4-2400 LPDDR3-2133
Intel® UHD Graphics	620	620	620	620
Graphics Dynamic Frequency (MHz)	Up to 1150	Up to 1150	Up to 1100	Up to 1100

Рисунок 3 – Базовые характеристики первых процессоров Core 8

Все новые мобильные процессоры Intel Core 8 поколения оснащены встроенным графическим ядром Intel UHD Graphics 620 с поддержкой до трех независимых дисплеев, унаследованным с некоторыми изменениями от процессоров 7 поколения (Kaby Lake, графика Intel HD Graphics 620).

Встроенная графика UHD Graphics 620 поддерживает кодеки HEVC и VP9, позволяет работать с 4K-видео с 10-битной глубиной цвета.

Раньше сложность CISC-инструкций была очень полезна, так как раньше ОЗУ была раньше медленной и небольшого объёма, то есть если компьютер мог исполнить что-то за одну инструкцию, то он использовал меньше оперативной памяти, чем при использовании нескольких, следовательно в памяти нужно было хранить меньше данных, что являлось несомненным плюсом.

На рисунке 4 изображена фотография кристалла чипа Intel Core 8 поколения.

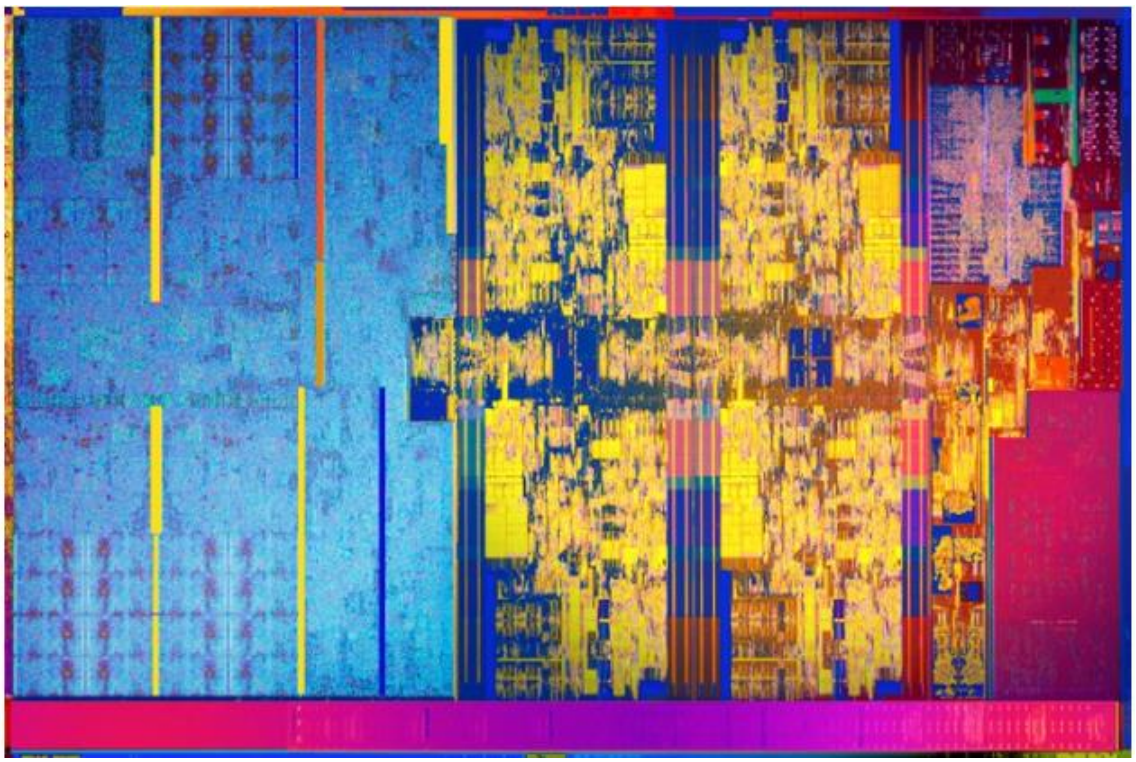


Рисунок 4 – Фотография кристалла чипа Intel Core 8 поколения

Если говорить про достоинства архитектуры x86, то сразу стоит упомянуть про обратную совместимость, что позволяет запустить программы intel 8086 на современных компьютерах, построенных на данной архитектуре. Также раньше использование CISC (complex instruction set computing) было неоспоримо. Здесь нельзя избежать сравнения с RISC (reduced instruction set computing) архитектурой. Если посмотреть на CISC, то там процессор исполняет больше работы в рамках одной инструкции, что приближает логику работы процессора к логике работы ПО, а RISC используют одну инструкцию в рамках одного действия. Раньше сложность CISC-инструкций была очень полезна, так как раньше ОЗУ была раньше медленной и небольшого объёма, то есть если компьютер мог исполнить что-то за одну инструкцию, то он использовал меньше оперативной памяти, чем при использовании нескольких, следовательно в памяти нужно было хранить меньше данных, что являлось несомненным плюсом. Недостаток здесь в том, что это преобразование требует дополнительной энергии, а следовательно и дополнительный отвод тепла.

1.3 Обоснование выбора вычислительной системы

Это производительный восьмиядерный чип, построенный на современной микроархитектуре, произведённый по 7нм техпроцессу, который сможет обеспечить должную производительность для выполнения курсовой работы.

1.4 Анализ выбранной вычислительной системы для написания программы

Вычислительная система состоит из:

– Накопителя: nvme SSD 256gb;

– Оперативной памяти: 16gb ddr4;

– Встроенной видеокарты: Nvidia GeForce 1060;

– Процессора: Intel Core i7-8750H.

На рисунке 5 представлена характеристика процессора с официального сайта производителя [3].

Частота	2200 - 4100 MHz
Кэш L1	384 KB
Кэш L2	1.5 MB
Кэш L3	9 MB
Количество ядер	6 / 12
Теплопакет (TDP)	45 Вт
Техпроцесс	14 нм
Макс. температура	100 °C
Разъём (сокет)	FCBGA1440
Особенности	Dual-Channel DDR4 Memory Controller, HyperThreading, AVX, AVX2, Quick Sync, Virtualization, AES-NI
Встроенная графика	Intel UHD Graphics 630 (350 - 1100 MHz)
64 бита	+
Architecture	x86
Дата анонса	03/03/2018 = 1678 days old

Рисунок 5 – Страница процессора на сайте производителя

На рисунке 6 представлена сравнительная характеристика различных процессоров с официального сайта производителя [11].

HWBOT x265 Benchmark v2.2 - HWBOT x265 4k Preset

min: 7.56 сред.: 8.2 медиана: 7.8 (17%) макс.: 9.55 fps

[-] 5 подробных результатов [+] Соседние видеопроцессоры

Модель	Процессор	Встроенная графика	RAM	Значение
MSI P65 8RF-451 - Creator	i7-8750H	GeForce GTX 1070 Max-Q 1266 / 8000 МГц 8 GB	16 GB	7.56
Schenker Work 15	i7-8750H	UHD Graphics 630 1100 МГц	16 GB	7.76
Schenker Work 15	i7-8750H	UHD Graphics 630 1100 МГц	16 GB	7.8
MSI GE75 8SG Raider	i7-8750H	GeForce RTX 2080 Mobile 1380 / 1750 МГц 8 GB	16 GB	8.43
Asus ROG Strix Scar II GL704GW	i7-8750H	GeForce RTX 2070 Mobile 1215 / 1750 МГц 8 GB	16 GB	9.55

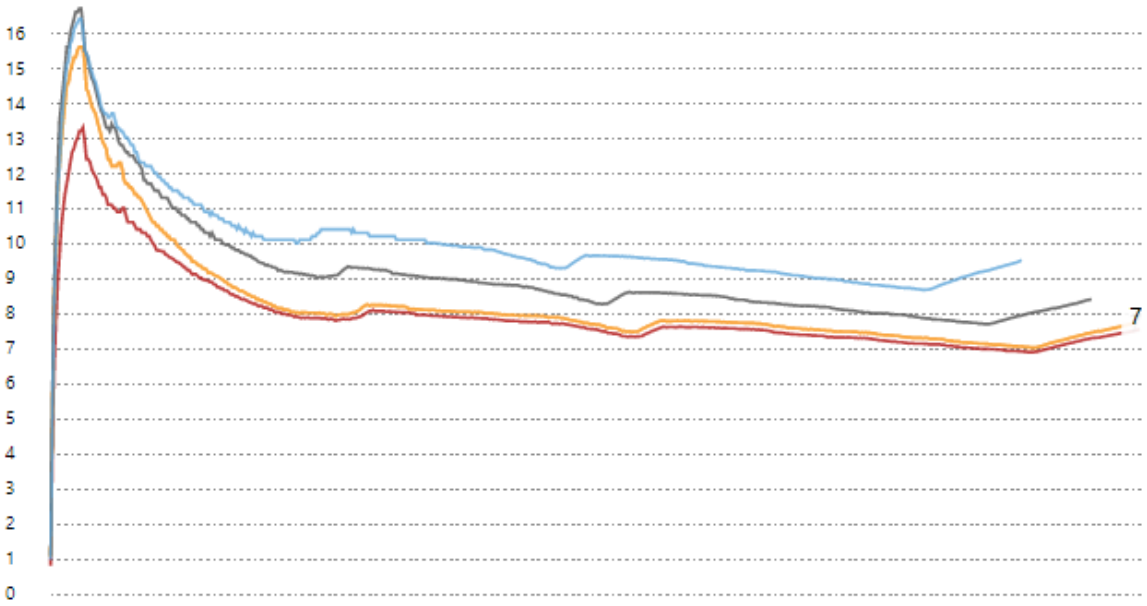


Рисунок 6 – Тесты на производительность HWBOT x265 Benchmark v2.2

2 ПЛАТФОРМА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

2.1 Структура и архитектура платформы

Архитектура **ОС Windows** претерпела ряд изменений в процессе эволюции. Первые версии системы имели микроядерный дизайн, основанный на микроядре Mach, которое было разработано в университете Карнеги-Меллона.

Архитектура более поздних версий системы микроядерной уже не является. Причина заключается в постепенном преодолении основного недостатка микроядерных архитектур – дополнительных накладных расходов, связанных с передачей сообщений. По мнению специалистов Microsoft, чисто микроядерный дизайн коммерчески невыгоден, поскольку неэффективен. Поэтому большой объем системного кода, в первую очередь управление системными вызовами и экранная графика, был перемещен из адресного пространства пользователя в пространство ядра и работает в привилегированном режиме. На рисунке 7 представлена упрощенная архитектурная схема операционной системы Windows [19].

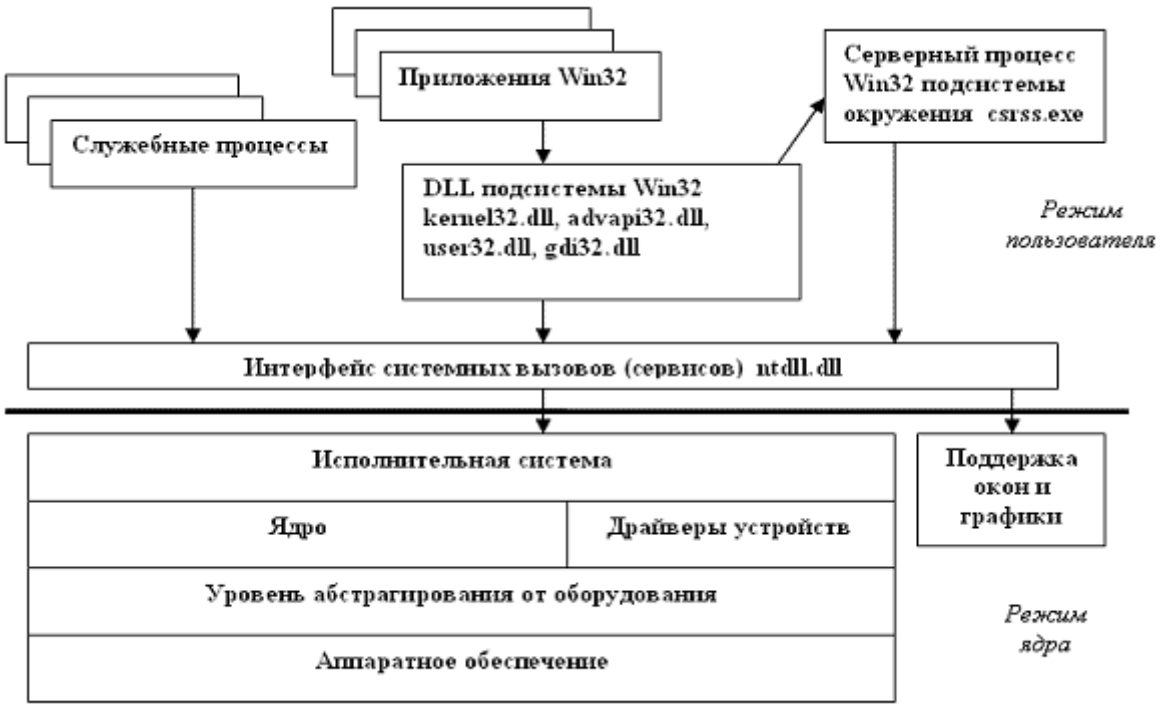


Рисунок 7 – Упрощенная архитектурная схема ОС Windows

Сегодня микроядро ОС Windows слишком велико (более 1 Мб), чтобы носить приставку "микро". Основные компоненты ядра Windows NT располагаются в вытесняемой памяти и взаимодействуют друг с другом путем передачи сообщений, как и положено в микроядерных операционных системах. В тоже время все компоненты ядра работают в одном адресном пространстве и активно используют общие структуры данных, что свойственно операционным системам с монолитным ядром.

Высокая модульность и гибкость первых версий Windows NT позволила успешно перенести систему на такие отличные от Intel платформы, как Alpha (корпорация DEC), Power PC (IBM) и MIPS (Silicon Graphic). Более поздние версии ограничиваются поддержкой архитектуры Intel x86.

2.2 Обоснование выбора операционной системы

Разработка приложения будет производиться с использованием библиотеки *windows.h* и иструментария Windows.

Набор функций для работы приложения, содержащийся в библиотеке *windows.h,* работающий под управлением ОС Windows – WIN32 API.

2.3 Анализ операционной системы

На рисунке 8 изображен анализ операционной системы, на которой производится разработка.

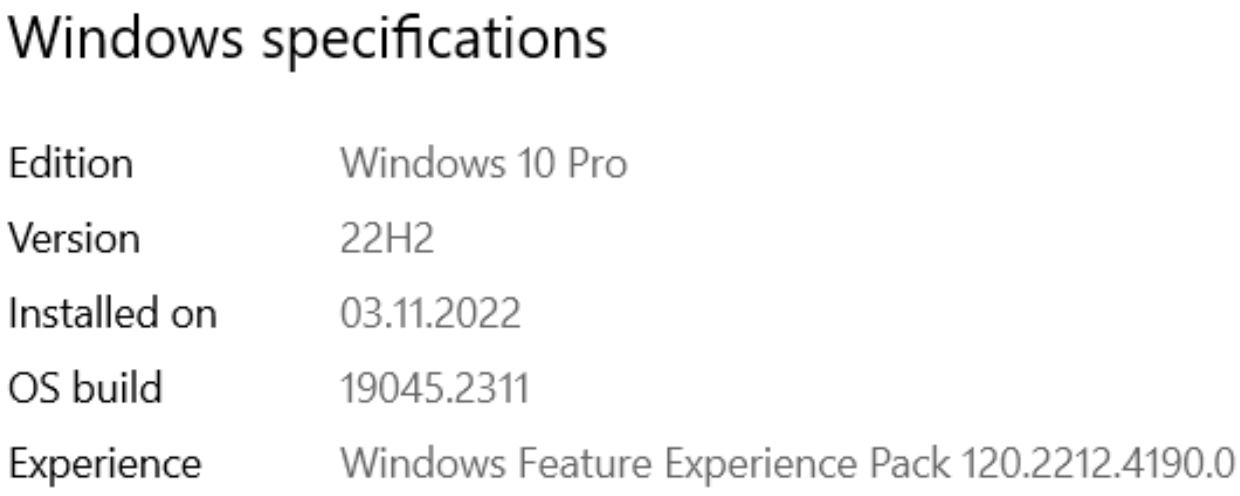


Рисунок 8 – Анализ операционной системы

3 ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА

3.1 Обоснование необходимости разработки

Перехват и протоколирование сетевого трафика являются важными задачами в области информационной безопасности. Эти задачи позволяют анализировать и отслеживать сетевой трафик, выявлять возможные уязвимости и атаки, а также собирать информацию о сетевых взаимодействиях. Для эффективного решения данных задач необходимо использовать специализированные инструменты, позволяющие перехватывать и анализировать сетевой трафик.

В данном проекте разрабатывается программа, предназначенная для перехвата и протоколирования сетевого трафика. Она позволяет отслеживать сетевые взаимодействия, анализировать передаваемые данные и определять протоколы, используемые в сети. Такая программа может быть использована для обнаружения возможных уязвимостей и атак, а также для анализа сетевого трафика в целях оптимизации работы сети.

3.2 Технологии программирования для решения задачи

Для разработки программного продукта были использованы следующие технологии:

– CSharp, язык программирования, используемый для написания приложения;

– .NET Framework - программная платформа, которая предоставляет API для работы с сетевыми устройствами;

– PcapDotNet - библиотека для работы с сетевым трафиком, которая предоставляет возможность перехвата и анализа сетевых пакетов;

– SharpPcap - библиотека для работы с сетевыми устройствами, которая предоставляет удобный интерфейс для работы с сетевыми интерфейсами.

Использование данных технологий позволило создать эффективный и удобный инструмент для перехвата и протоколирования сетевого трафика.

C# (произносится "Си шарп") – это объектно-ориентированный язык программирования, разработанный компанией Microsoft. C# был создан в 2000 году как часть платформы .NET Framework, и с тех пор стал одним из самых

популярных языков программирования для создания приложений на платформе Windows.

Язык объединяет в себе преимущества C++ и Java, а также содержит ряд новых возможностей, таких как поддержка делегатов и атрибутов. Он поддерживает многопоточное программирование, асинхронные операции, обработку исключений и другие функции, необходимые для разработки сложных приложений.

C# компилируется в промежуточный язык (IL), который затем исполняется виртуальной машиной Common Language Runtime (CLR). Это позволяет разрабатывать приложения на C#, которые могут быть запущены на различных платформах, поддерживающих .NET Framework.

Язык имеет богатую стандартную библиотеку классов, которая включает в себя классы для работы с сетью, вводом-выводом, коллекциями, графическими интерфейсами пользователя и многими другими функциями.

C# используется для создания различных типов приложений, включая настольные приложения, веб-приложения, игры и мобильные приложения. Он также используется для написания скриптов и автоматизации задач в Unity3D.

.NET Framework – это программная платформа от компании Microsoft, предназначенная для разработки и выполнения приложений на языке программирования C#. Эта платформа включает в себя среду выполнения приложений, базовую библиотеку классов, компиляторы и другие инструменты, которые необходимы для разработки приложений под Windows.

Среди прочих возможностей, .NET Framework также предоставляет API для работы с сетевыми устройствами, такими как сетевые интерфейсы и пакеты. Это API включает в себя различные библиотеки, которые могут использоваться для создания сетевых приложений, таких как мониторинг сетевого трафика или создание сетевых утилит.

Для работы с сетевыми устройствами на языке C# можно использовать библиотеки, такие как SharpPcap или PcapDotNet, которые предоставляют удобный и гибкий доступ к сетевым интерфейсам и пакетам. Эти библиотеки включают в себя множество функций для обработки и анализа пакетов, а также возможности для записи и воспроизведения сетевого трафика.

PcapDotNet – это библиотека для работы с сетевым трафиком, предназначенная для перехвата, анализа и создания сетевых пакетов. Библиотека используется для решения задач сетевого администрирования, тестирования безопасности сети и разработки сетевых приложений.

PcapDotNet поддерживает множество протоколов и форматов данных, включая Ethernet, IPv4, IPv6, TCP, UDP, DNS и другие. Библиотека позволяет перехватывать и анализировать сетевой трафик на основе различных критериев, таких как источник, назначение, тип протокола, порт назначения и т.д. Кроме того, PcapDotNet может записывать перехваченные пакеты в файлы, что позволяет анализировать трафик в последующем.

PcapDotNet доступна на языке C# и является частью платформы .NET Framework. Библиотека предоставляет простой и удобный API для работы с сетевым трафиком и может использоваться как для создания собственных сетевых приложений, так и для интеграции с другими приложениями.

SharpPcap – это библиотека для работы с сетевыми устройствами в операционной системе Windows, которая предоставляет простой и удобный интерфейс для работы с сетевыми интерфейсами. Она позволяет перехватывать и анализировать сетевой трафик на выбранных сетевых интерфейсах, получать информацию о сетевых интерфейсах, а также управлять сетевыми интерфейсами. Библиотека SharpPcap основана на библиотеке WinPcap и предоставляет более высокоуровневый интерфейс для работы с сетевыми устройствами.

4 ПРОЕКТИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ ВОЗМОЖНОСТЕЙ ПРОГРАММЫ

Интерфейс приложения выполнен в соответствии с требованиями, которые предъявлялись к системе на этапе проектирования. Простота и удобство использования приложения обеспечивает выполнение качественной и многофункциональной работы без напряжения и без затрат времени на осмысление информации, которая отображается на экране.

Основной особенностью прикладного программного обеспечения является наличие интерфейса пользователя, не требующего специальных навыков работы с ПЭВМ. Все это находит практическую реализацию в разработанном программном продукте.

Простота интерфейса приложения подразумевает, что не нужно усложнять восприятие и понимание информации, возникающей перед глазами пользователя. Для этого при выполнении очередного шага задания перед пользователем будет отображаться минимально необходимая информация.

Также были исключены многословные командные имёна и сообщения и элементы управления размещены на экране с учётом их смыслового значения и логической взаимосвязи. При запуске программы открывается главное окно приложения.

Интерфейс предоставляет исключительно ознакомительную информацию, без возможности влиять на содержимое или изменять значения отображаемых атрибутов. Осуществляется постоянное обновление данных, что позволяет отслеживать и, в случае необходимости, проводить диагностику.

5 АРХИТЕКТУРА РАЗРАБАТЫВАЕМОЙ ПРОГРАММЫ

5.1 Общая структура программы

Архитектура разрабатываемой программы основана на принципах объектно-ориентированного программирования и состоит из нескольких основных компонентов.

Основная структура программы включает в себя:

– Пользовательский интерфейс, который отображает информацию о доступных сетевых интерфейсах, позволяет выбрать интерфейс для захвата сетевого трафика и отображает информацию о перехваченных пакетах.

– Модуль захвата трафика, который осуществляет перехват и анализ сетевых пакетов, используя библиотеки PcapDotNet и SharpPcap.

– Модуль анализа сетевых пакетов, который обрабатывает перехваченные пакеты и извлекает из них необходимую информацию, такую как адреса источника и назначения, тип протокола и размер пакета.

– Модуль записи перехваченных пакетов в файл для дальнейшего анализа.

Каждый компонент программы имеет свою функциональность и может работать независимо друг от друга. Однако, вместе они обеспечивают полноценный механизм перехвата и анализа сетевого трафика.

5.2 Описание функциональной схемы программы

Функциональная схема программы включает в себя следующие блоки:

– Интерфейс пользователя – блок, отвечающий за взаимодействие пользователя с программой. Включает в себя элементы управления, такие как кнопки, поля ввода, таблицы и т.д.

– Блок управления сетевым интерфейсом – блок, который позволяет выбрать и настроить сетевой интерфейс, с которым будет работать программа.

– Блок захвата пакетов – блок, который осуществляет захват сетевых пакетов с выбранного сетевого интерфейса и сохраняет их в память или файл.

– Блок анализа пакетов – блок, который осуществляет анализ захваченных сетевых пакетов и извлекает из них информацию, такую как исходный и конечный IP-адреса, протокол, тип и содержимое пакета.

– Блок отображения результатов – блок, который отображает результаты анализа сетевых пакетов в удобном для пользователя формате, например, в таблице или графическом виде.

– Блок сохранения результатов – блок, который позволяет сохранить результаты анализа сетевых пакетов в файл для дальнейшего использования.

Функциональная схема программы показывает, какие блоки взаимодействуют между собой и какие функции выполняются каждым из них. Она помогает понять, как устроена программа и какие ее возможности.

5.3 Описание блок-схемы алгоритма программы

ЗАКЛЮЧЕНИЕ

Курсовой проект представляет собой приложение для захвата и анализа сетевого трафика с использованием библиотеки SharpPcap в среде разработки Microsoft Visual Studio с помощью языка программирования C#.

Приложение предоставляет возможность выбора сетевого интерфейса, начала и остановки захвата трафика, анализа захваченных пакетов и отображения их в удобной для пользователя форме, а также сохранения захваченных пакетов в файл.

В результате работы были достигнуты поставленные цели, а именно изучение сетевых протоколов, использование библиотеки SharpPcap для захвата и анализа сетевого трафика, а также разработка приложения на языке программирования C#.

В процессе выполнения курсового проекта были получены навыки программирования на C#, работа с библиотекой SharpPcap, изучение сетевых протоколов, а также организация работы в команде и планирование проекта.

В дальнейшем, приложение может быть доработано и расширено, добавлены новые функции, например, фильтрацию пакетов по определенным критериям, анализ сетевого трафика в реальном времени и многие другие.

СПИСОК ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

[1] Web Znam [Электронный ресурс]. – Развитие процессоров Intel Core. – Режим доступа : https://webznam.ru/blog/razvitie_processorov_intel_core/2021-12-23-1894/. – Дата доступа: 29.10.2022.

[2] Notebookcheck [Электронный ресурс]. – Intel Core i7-8750H. – Режим доступа : https://www.notebookcheck-ru.com/Intel-Core-i7-8750H.331378.0.html. – Дата доступа: 03.11.2022.

[3] Intel [Электронный ресурс]. – Intel® Core™ i7-8750H Processor. – Режим доступа: https://ark.intel.com/content/www/us/en/ark/products/134906/intel-core-i78750h-processor-9m-cache-up-to-4-10-ghz.html?q=8750h. – Дата доступа: 01.11.2022.

[4] HWBOT Benchmark [Электронный ресурс]. – Инструментарий управления Windows (WMI). – Режим доступа : https://hwbot.org/benchmarks. – Дата доступа: 03.11.2022.

[5] НОУ ИНТУИНТ [Электронный ресурс]. – Лекция 1: Создание ОС Windows. Структура ОС Windows – Режим доступа : https://intuit.ru/studies/professional_retraining/962/courses/217/lecture/5585?page=2 – Дата доступа: 12.11.2022.

[6] MSDN [Электронный ресурс]. – Сведения о WMI. – Режим доступа : https://learn.microsoft.com/ru-ru/windows/win32/wmisdk/about-wmi?source=recommendations. – Дата доступа: 14.11.2022.

[7] MSDN [Электронный ресурс]. – Библиотека COM. – Режим доступа : https://learn.microsoft.com/ru-ru/windows/win32/com/the-com-library. – Дата доступа: 01.11.2022.

[8] MSDN [Электронный ресурс]. – Инструментарий управления Windows (WMI). – Режим доступа : https://learn.microsoft.com/ru-ru/windows/win32/wmisdk/wmi-start-page. – Дата доступа: 14.11.2022.

[9] MSDN [Электронный ресурс]. – Архитектура WMI. – Режим доступа : https://learn.microsoft.com/ru-ru/windows/win32/wmisdk/wmi-architecture. – Дата доступа: 16.11.2022

[10] PPT Online [Электронный ресурс]. – Системное программирование – Режим доступа : https://ppt-online.org/505837. – Дата доступа: 16.11.2022.

[11] MSDN [Электронный ресурс]. – Создание подключения к пространству имен WMI. – Режим доступа : https://learn.microsoft.com/ru-ru/windows/win32/wmisdk/creating-a-connection-to-a-wmi-namespace. – Дата доступа: 16.11.2022.

[12] MSDN [Электронный ресурс]. – Сообщение WM_TIMER. – Режим доступа : https://learn.microsoft.com/ru-ru/windows/win32/winmsg/wm-timer. – Дата доступа: 01.11.2022. – Дата доступа: 21.11.2022.

[13] Frolov-lib.ru [Электронный ресурс]. – Операционная система Microsoft Windows 3.1 для программиста. – Режим доступа : https://www.frolov-lib.ru/books/bsp/v11/ch1_2.htm. – Дата доступа: 22.11.2022.

[14] Первые шаги [Электронный ресурс]. – Шаг 42 - Функция WinMain() – Режим доступа : https://firststeps.ru/mfc/winapi/r.php?42. – Дата доступа: 22.11.2022.

[15] GitHub [Электронный ресурс]. – RRUZ | Wmi delphi code creator. – Режим доступа : https://github.com/RRUZ/wmi-delphi-code-creator. – Дата доступа: 23.11.2022.

[16] StudFile [Электронный ресурс]. – Файловый архив студентов | Операционные системы. – Режим доступа : https://studfile.net/preview/11218660. – Дата доступа: 29.11.2022.

[17] Литвиенко, Н.А. Технология программирования на C++. Win32 API-приложения. : Учеб. пособие / Н.А. Литвинко. – Санкт-Петербург : ИТМО, 2010. – 281 с.

[18] AIDA64 [Электронный ресурс]. – AIDA64 Extreme – Режим доступа : https://aida64russia.com/. – Дата доступа: 05.12.2022.

[19] Первые шаги [Электронный ресурс]. – Шаг 56 - Основная функция окна – Режим доступа : https://firststeps.ru/mfc/winapi/r.php?56. – Дата доступа: 05.12.2022.

[20] Integra Sources [Электронный ресурс]. – Technology Overview on Software Development for Battery Management Systems (BMS). – Режим доступа : https://www.integrasources.com/blog/battery-management-systems-software-development/. – Дата доступа: 07.12.2022.

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг программного кода

```
using System;
using System.IO;
using System.Threading;
using System.Collections.Generic;
using System.ComponentModel;
using System.Windows.Forms;
using SharpPcap.LibPcap;
using SharpPcap;
using PacketDotNet;


namespace pcapTest
{
    public partial class MainForm : Form
    {
        List<LibPcapLiveDevice> interfaceList = new List<LibPcapLiveDevice>();
        int selectedIntIndex;
        LibPcapLiveDevice wifi_device;
        CaptureFileWriterDevice captureFileWriter;
        Dictionary<int, Packet> capturedPackets_list = new Dictionary<int,
Packet>();

        int packetNumber = 1;
        string time_str = "", sourceIP = "", destinationIP = "", protocol_type =
"", length = "";

        bool startCapturingAgain = false;

        Thread sniffing;

        public MainForm(List<LibPcapLiveDevice> interfaces, int selectedIndex)
        {
            InitializeComponent();
            this.interfaceList = interfaces;
            selectedIntIndex = selectedIndex;
            // Extract a device from the list
            wifi_device = interfaceList[selectedIntIndex];
        }

        private void MainForm_FormClosed(object sender, FormClosedEventArgs e)
        {
            System.Windows.Forms.Application.Exit();
```

```
        }

        private void toolStripButton1_Click(object sender, EventArgs e)// Start
sniffing
        {
            if(startCapturingAgain == false) //first time
            {
                System.IO.File.Delete(Environment.CurrentDirectory +
"capture.pcap");
                wifi_device.OnPacketArrival += new
PacketArrivalEventHandler(Device_OnPacketArrival);
                sniffing = new Thread(new ThreadStart(sniffing_Proccess));
                sniffing.Start();
                toolStripButton1.Enabled = false;
                toolStripButton2.Enabled = true;
                textBox1.Enabled = false;

            }
           else if (startCapturingAgain)
            {
                if (MessageBox.Show("Your packets are captured in a file.
Starting a new capture will override existing ones.", "Confirm",
MessageBoxButtons.OK, MessageBoxIcon.Warning) == DialogResult.OK)
                {
                    // user clicked ok
                    System.IO.File.Delete(Environment.CurrentDirectory +
"capture.pcap");
                    listView1.Items.Clear();
                    capturedPackets_list.Clear();
                    packetNumber = 1;
                    textBox2.Text = "";
                    wifi_device.OnPacketArrival += new
PacketArrivalEventHandler(Device_OnPacketArrival);
                    sniffing = new Thread(new ThreadStart(sniffing_Proccess));
                    sniffing.Start();
                    toolStripButton1.Enabled = false;
                    toolStripButton2.Enabled = true;
                    textBox1.Enabled = false;
                }
            }
            startCapturingAgain = true;
        }

        // paket information
        private void listView1_ItemSelectionChanged(object sender,
ListViewItemSelectionChangedEventArgs e)
        {
            string protocol = e.Item.SubItems[4].Text;
            int key = Int32.Parse(e.Item.SubItems[0].Text);
```

```
            Packet packet;
            bool getPacket  = capturedPackets_list.TryGetValue(key, out packet);

                switch (protocol) {
                case "TCP":
                    if(getPacket)
                    {
                        var tcpPacket =
(TcpPacket)packet.Extract(typeof(TcpPacket));
                        if (tcpPacket != null)
                        {
                            int srcPort = tcpPacket.SourcePort;
                            int dstPort = tcpPacket.DestinationPort;
                            var checksum = tcpPacket.Checksum;

                            textBox2.Text = "";
                            textBox2.Text = "Packet number: " + key +
                                            " Type: TCP" +
                                            "\r\nSource port:" + srcPort +
                                            "\r\nDestination port: " + dstPort +
                                            "\r\nTCP header size: " +
tcpPacket.DataOffset +
                                            "\r\nWindow size: " +
tcpPacket.WindowSize + // bytes that the receiver is willing to receive
                                            "\r\nChecksum:" + checksum.ToString()
+ (tcpPacket.ValidChecksum ? ",valid" : ",invalid") +
                                            "\r\nTCP checksum: " +
(tcpPacket.ValidTCPChecksum ? ",valid" : ",invalid") +
                                            "\r\nSequence number: " +
tcpPacket.SequenceNumber.ToString() +
                                            "\r\nAcknowledgment number: " +
tcpPacket.AcknowledgmentNumber + (tcpPacket.Ack ? ",valid" : ",invalid") +
                                            // flags
                                            "\r\nUrgent pointer: " +
(tcpPacket.Urg ? "valid" : "invalid") +
                                            "\r\nACK flag: " + (tcpPacket.Ack ?
"1" : "0") + // indicates if the AcknowledgmentNumber is valid
                                            "\r\nPSH flag: " + (tcpPacket.Psh ?
"1" : "0") + // push 1 = the receiver should pass the data to the app immidiatly,
don't buffer it
                                            "\r\nRST flag: " + (tcpPacket.Rst ?
"1" : "0") + // reset 1 is to abort existing connection
                                            // SYN indicates the sequence numbers
should be synchronized between the sender and receiver to initiate a connection
                                            "\r\nSYN flag: " + (tcpPacket.Syn ?
"1" : "0") +
                                            // closing the connection with a
deal, host_A sends FIN to host_B, B responds with ACK
```

```
                                                // FIN flag indicates the sender is
finished sending
                                                "\r\nFIN flag: " + (tcpPacket.Fin ?
"1" : "0") +
                                                "\r\nECN flag: " + (tcpPacket.ECN ?
"1" : "0") +
                                                "\r\nCWR flag: " + (tcpPacket.CWR ?
"1" : "0") +
                                                "\r\nNS flag: " + (tcpPacket.NS ? "1"
: "0");
                        }
                    }
                    break;
                case "UDP":
                    if (getPacket)
                    {
                        var udpPacket =
(UdpPacket)packet.Extract(typeof(UdpPacket));
                        if (udpPacket != null)
                        {
                            int srcPort = udpPacket.SourcePort;
                            int dstPort = udpPacket.DestinationPort;
                            var checksum = udpPacket.Checksum;

                            textBox2.Text = "";
                            textBox2.Text = "Packet number: " + key +
                                            " Type: UDP" +
                                            "\r\nSource port:" + srcPort +
                                            "\r\nDestination port: " + dstPort +
                                            "\r\nChecksum:" + checksum.ToString()
+ " valid: " + udpPacket.ValidChecksum +
                                            "\r\nValid UDP checksum: " +
udpPacket.ValidUDPChecksum;
                        }
                    }
                    break;
                case "ARP":
                    if (getPacket)
                    {
                        var arpPacket =
(ARPPacket)packet.Extract(typeof(ARPPacket));
                        if (arpPacket != null)
                        {
                            System.Net.IPAddress senderAddress =
arpPacket.SenderProtocolAddress;
                            System.Net.IPAddress targerAddress =
arpPacket.TargetProtocolAddress;
                            System.Net.NetworkInformation.PhysicalAddress
senderHardwareAddress = arpPacket.SenderHardwareAddress;
```

```
                            System.Net.NetworkInformation.PhysicalAddress
targerHardwareAddress = arpPacket.TargetHardwareAddress;

                            textBox2.Text = "";
                            textBox2.Text = "Packet number: " + key +
                                            " Type: ARP" +
                                            "\r\nHardware address length:" +
arpPacket.HardwareAddressLength +
                                            "\r\nProtocol address length: " +
arpPacket.ProtocolAddressLength +
                                            "\r\nOperation: " +
arpPacket.Operation.ToString() + // ARP request or ARP reply ARP_OP_REQ_CODE,
ARP_OP_REP_CODE
                                            "\r\nSender protocol address: " +
senderAddress +
                                            "\r\nTarget protocol address: " +
targerAddress +
                                            "\r\nSender hardware address: " +
senderHardwareAddress +
                                            "\r\nTarget hardware address: " +
targerHardwareAddress;
                        }
                    }
                    break;
                case "ICMP":
                    if (getPacket)
                    {
                        var icmpPacket =
(ICMPv4Packet)packet.Extract(typeof(ICMPv4Packet));
                        if (icmpPacket   != null)
                        {
                            textBox2.Text = "";
                            textBox2.Text = "Packet number: " + key +
                                            " Type: ICMP v4" +
                                            "\r\nType Code: 0x" +
icmpPacket.TypeCode.ToString("x") +
                                            "\r\nChecksum: " +
icmpPacket.Checksum.ToString("x") +
                                            "\r\nID: 0x" +
icmpPacket.ID.ToString("x") +
                                            "\r\nSequence number: " +
icmpPacket.Sequence.ToString("x");
                        }
                    }
                    break;
                case "IGMP":
                    if (getPacket)
                    {
```

```
                        var igmpPacket =
(IGMPv2Packet)packet.Extract(typeof(IGMPv2Packet));
                        if (igmpPacket != null)
                        {
                            textBox2.Text = "";
                            textBox2.Text = "Packet number: " + key +
                                            " Type: IGMP v2" +
                                            "\r\nType: " + igmpPacket.Type +
                                            "\r\nGroup address: " +
igmpPacket.GroupAddress +
                                            "\r\nMax response time" +
igmpPacket.MaxResponseTime;
                        }
                    }
                    break;
                default:
                    textBox2.Text = "";
                    break;
                }
        }

        private void toolStripButton6_Click(object sender, EventArgs e)// last
packet
        {
            var items = listView1.Items;
            var last = items[items.Count - 1];
            last.EnsureVisible();
            last.Selected = true;
        }

        private void toolStripButton5_Click(object sender, EventArgs e)// fist
packet
        {
            var first = listView1.Items[0];
            first.EnsureVisible();
            first.Selected = true;
        }

        private void toolStripButton4_Click(object sender, EventArgs e)//next
        {
            if(listView1.SelectedItems.Count == 1)
            {
                int index = listView1.SelectedItems[0].Index;
                listView1.Items[index + 1].Selected = true;
                listView1.Items[index + 1].EnsureVisible();
            }
        }
```

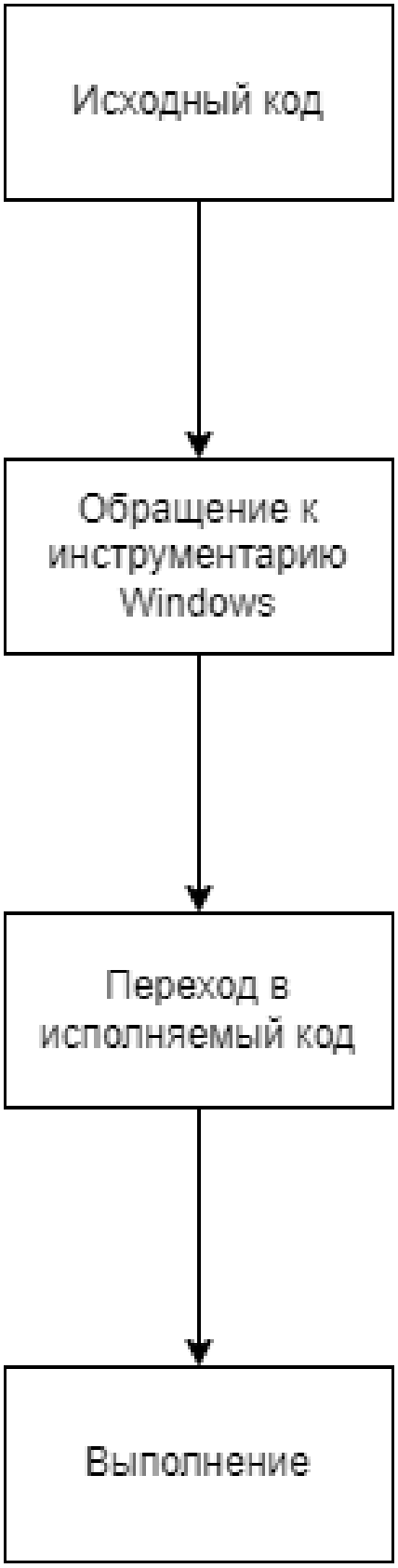
```
        private void chooseInterfaceToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            Interfaces openInterfaceForm = new Interfaces();
            this.Hide();
            openInterfaceForm.Show();
        }

        private void toolStripButton3_Click(object sender, EventArgs e)// prev
        {
            if (listView1.SelectedItems.Count == 1)
            {
                int index = listView1.SelectedItems[0].Index;
                listView1.Items[index - 1].Selected = true;
                listView1.Items[index - 1].EnsureVisible();
            }
        }

        private void toolStripButton2_Click(object sender, EventArgs e)// Stop
sniffing
        {
            sniffing.Abort();
            wifi_device.StopCapture();
            wifi_device.Close();
            captureFileWriter.Close();

            toolStripButton1.Enabled = true;
            textBox1.Enabled = true;
            toolStripButton2.Enabled = false;
        }

        private void sniffing_Proccess()
        {
            // Open the device for capturing
            int readTimeoutMilliseconds = 1000;
            wifi_device.Open(DeviceMode.Promiscuous, readTimeoutMilliseconds);

            // Start the capturing process
            if (wifi_device.Opened)
            {
                if (textBox1.Text != "")
                {
                    wifi_device.Filter = textBox1.Text;
                }
                captureFileWriter = new CaptureFileWriterDevice(wifi_device,
Environment.CurrentDirectory + "capture.pcap");
                wifi_device.Capture();
            }
        }
```

```
        public void Device_OnPacketArrival(object sender, CaptureEventArgs e)
        {
            // dump to a file
            captureFileWriter.Write(e.Packet);


            // start extracting properties for the listview
            DateTime time = e.Packet.Timeval.Date;
                time_str = (time.Hour + 1 ) + ":" + time.Minute + ":" +
time.Second + ":" + time.Millisecond;
                length = e.Packet.Data.Length.ToString();
                var packet =
PacketDotNet.Packet.ParsePacket(e.Packet.LinkLayerType, e.Packet.Data);

                // add to the list
                capturedPackets_list.Add(packetNumber, packet);


            var ipPacket = (IpPacket)packet.Extract(typeof(IpPacket));


                if (ipPacket != null)
                {
                    System.Net.IPAddress srcIp = ipPacket.SourceAddress;
                    System.Net.IPAddress dstIp = ipPacket.DestinationAddress;
                    protocol_type = ipPacket.Protocol.ToString();
                    sourceIP = srcIp.ToString();
                    destinationIP = dstIp.ToString();



                    var protocolPacket = ipPacket.PayloadPacket;

                    ListViewItem item = new
ListViewItem(packetNumber.ToString());
                    item.SubItems.Add(time_str);
                    item.SubItems.Add(sourceIP);
                    item.SubItems.Add(destinationIP);
                    item.SubItems.Add(protocol_type);
                    item.SubItems.Add(length);
                    Action action = () => listView1.Items.Add(item);
                    listView1.Invoke(action);

                    ++packetNumber;
                }
        }
    }
}
```

ПРИЛОЖЕНИЕ Б

(обязательное)

Функциональная схема программы

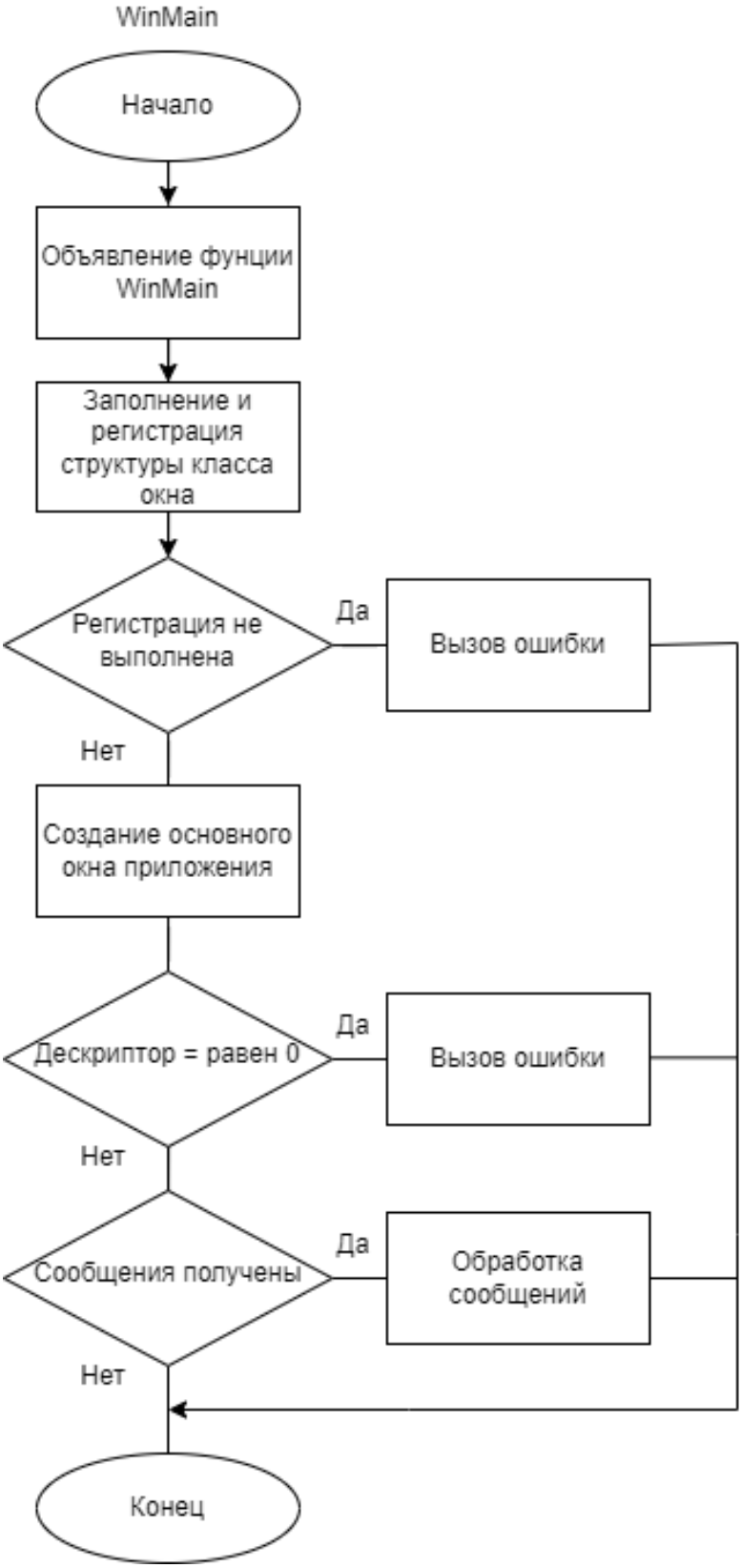


					ГУИР 053504.016.01 ПЛ	
Из	Лис	№ докум	Подпись	Дата		
Разраб.		Матвеев			Функциональная схема алгоритма	Литера: Т; Лист: 1; Листов: 1
Пров.		Калиновская				
Рец.						Кафедра информатики группа 053504
Н.контр.		Калиновская				
Утв.		Марков				

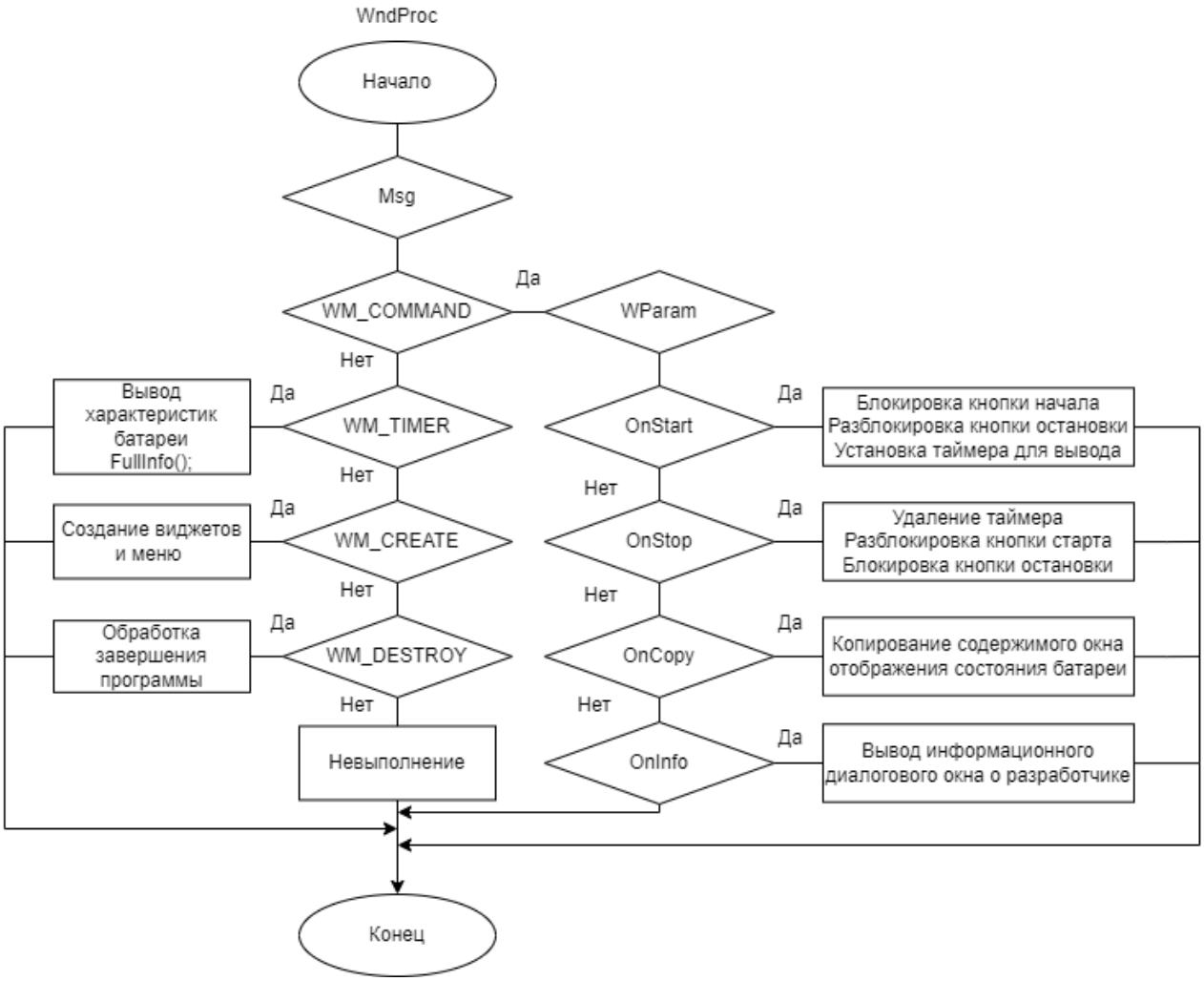
ПРИЛОЖЕНИЕ В

(обязательное)

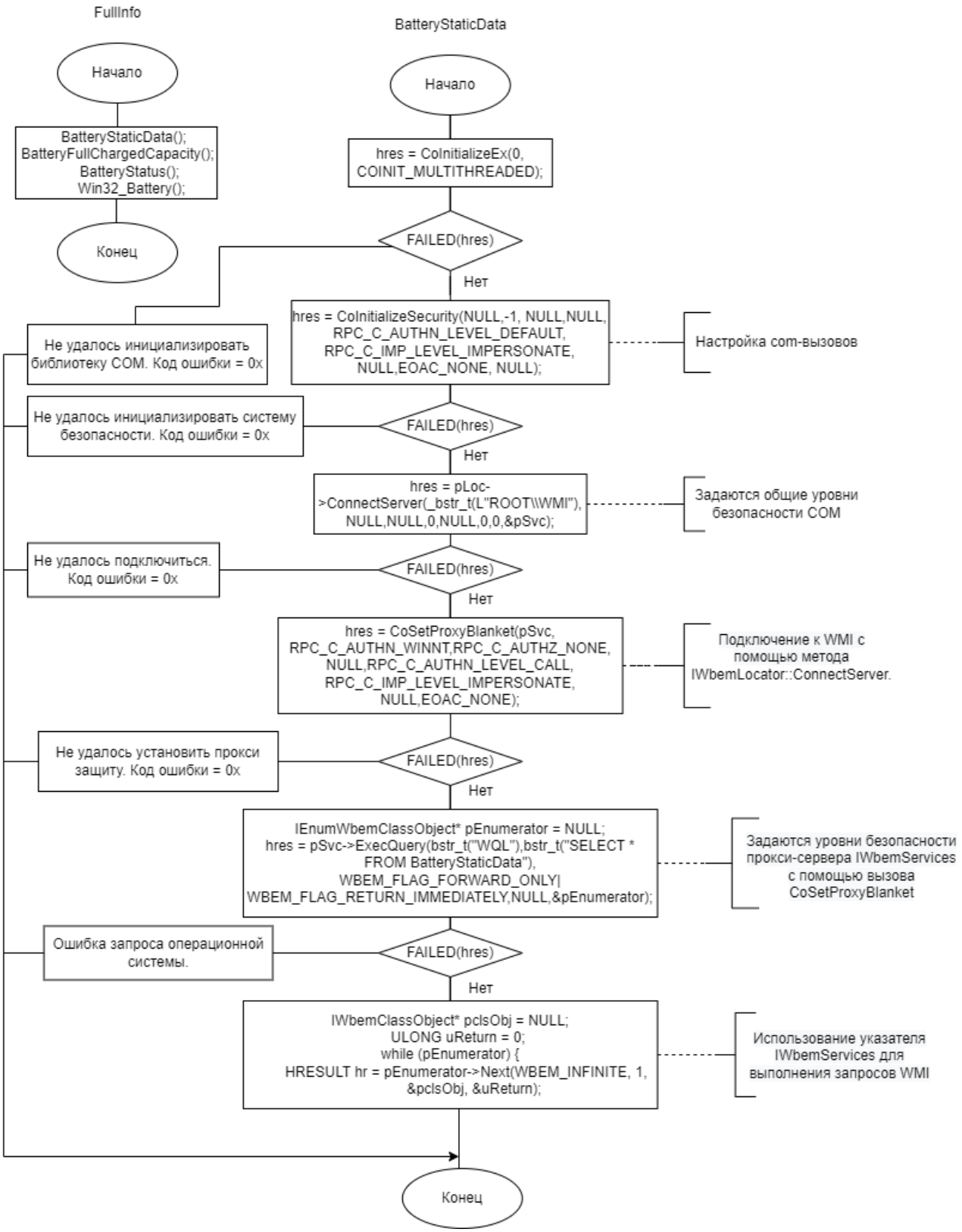
Блок-схема алгоритма программы



					ГУИР 053504.016.02 ПЛ	
Из	Лис	№ докум	Подпись	Дат		
Разраб.		Матвеев			Блок-схема алгоритма	Литера: Т; Лист: 1; Листов: 3
Пров.		Калиновская				
Рец.						Кафедра информатики группа 053504
Н.контр.		Калиновская				
Утв.		Марков				



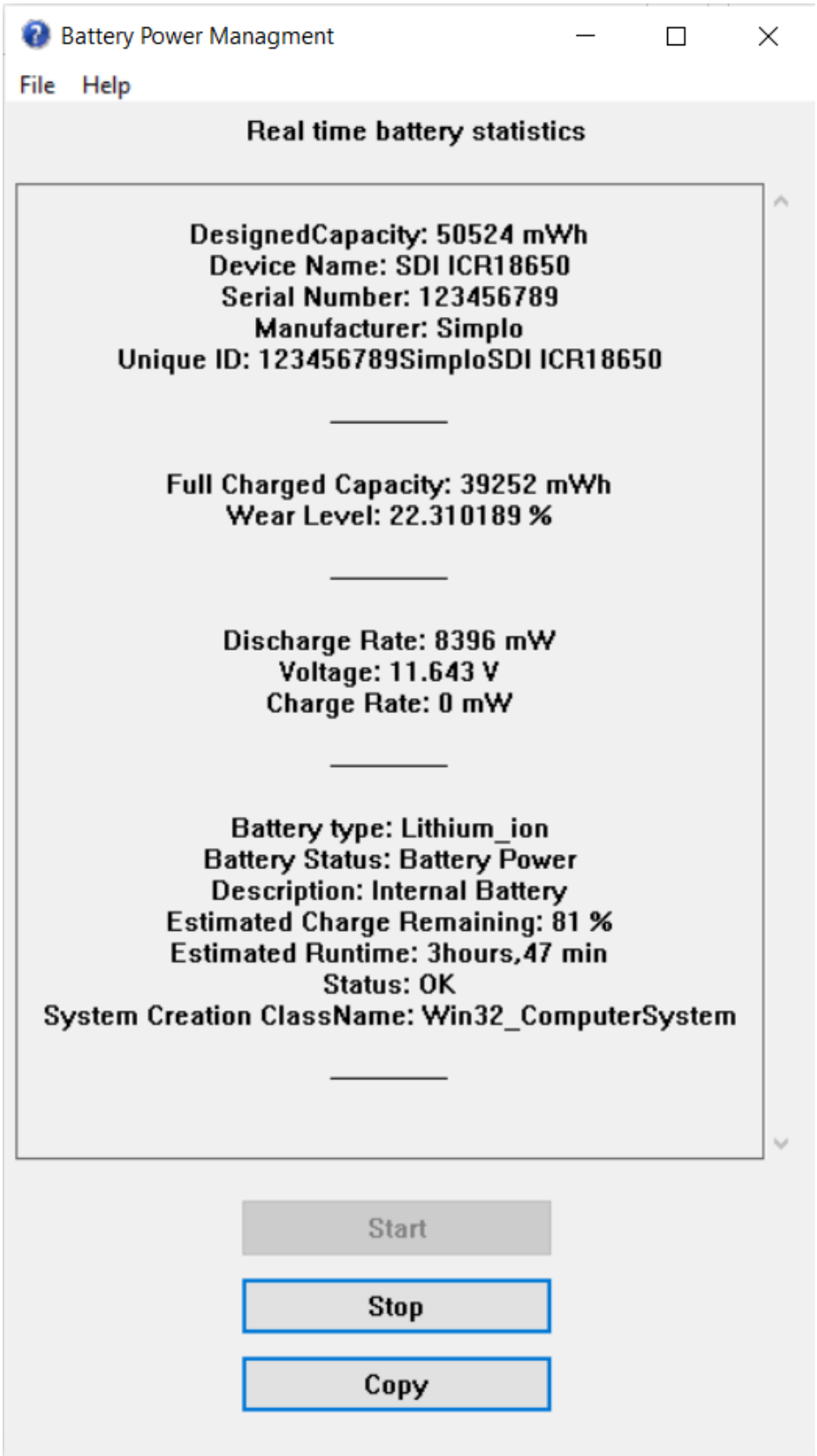
					ГУИР 053504.016.02 ПЛ	
Из	Лис	№ докум	Подпись	Дат		
Разраб.		Матвеев			Блок-схема алгоритма	Литера: Т; Лист: 2; Листов: 3
Пров.		Калиновская				
Рец.						Кафедра информатики группа 053504
Н.контр.		Калиновская				
Утв.		Марков				



					ГУИР 053504.016.02 ПЛ		
Из	Лис	№ докум	Подпись	Дат			
Разраб.		Матвеев			Блок-схема алгоритма	Литера: Т	Лист: 3 / Листов: 3
Пров.		Калиновская					
Рец.						Кафедра информатики группа 053504	
Н.контр.		Калиновская					
Утв.		Марков					

ПРИЛОЖЕНИЕ Г
(обязательное)

Графический интерфейс пользователя



					ГУИР 053504.016.02 ПЛ	
Из	Лис	№ докум	Подпись	Дат		
Разраб.		Матвеев			Графический интерфейс пользователя	Литера: Т; Лист: ; Листов: 1
Пров.		Калиновская				
Рец.						Кафедра 38 информатики группа 053504
Н.контр.		Калиновская				
Утв.		Марков				

ПРИЛОЖЕНИЕ Д
(обязательное)

Ведомость