Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Операционные системы и системное программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

ДРАЙВЕР УСТРОЙСТВАБГУИР КП 1-40 02 01 427 ПЗ

|  |  |  |
| --- | --- | --- |
| Студент |  | Группы 150504  А.С. Шалаш |
| Руководитель |  | Д.В. Басак |

Минск 2023

Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ЭВМ

*––––––––––––––––––––––––*

(подпись)

–––––––––––––––––––2023 г.

ЗАДАНИЕ

по курсовому проектированию

Студенту    *Шалаш Анне Сергеевне––––––––– ––––––––– –*

1. Тема проекта   *Драйвер устройства* –––––– –– ––––

2. Срок сдачи студентом законченного проекта –––––*17 мая 2023 г.– —––*

3. Содержание расчетно-пояснительной записки (список вопросов, которые подлежат развитию)

*Введение. 1. Обзор литературы. 2. Структурное проектирование. 3. Функциональное проектирование. 4. Инструкция пользователя. Заключение. Список используемых источников. Приложение А. Приложение Б. Приложение В. Приложение Г.*

5. Список графического материяла:

*1. Скриншоты работы программы –––––––––––––––––––––––– ––––*

*2. Схема основных алгоритмов программы. –––––––––––––––––––––– ––––––*

*3. Ведомость документов. . –––––––––––––––––––––––––––––*

6. Консультант по проекту   *Д.В. Басак ––––––––––––––––––––––––– ––– –*

7. Дата выдачи задания –––––*16 февраля 2023 г. –––––––––––––––––– ––*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

*раздзел 1 до 1.3 – 20 %;––––– –––––––––––––––––––––––––– ––––––*

*разделы 2, 3 до 1.4 – 40 %;– ––––––––– ––––––––––––––––––––––––––*

*раздел 4 до 15.4 – 20 %;––– –––––––––– –––––––––––––––––––––––––– –*

*оформление пояснительной записки и графического материала до 1.5 – 20%*

*Защита курсового проекта с 29 мая по 9 июня–––––––––––––––––––––––––––*

РУКОВОДИТЕЛЬ*– ––––––––––– Д.В. Басак*

(подпись)

Задание принял к исполнению*–––––––\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_–– А. С. Шалаш*

(дата и подпись студента)

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc136851841)

[1 ОБЗОР ЛИТЕРАТУРЫ 7](#_Toc136851842)

[1.1 Реализация с использованием WDK 8](#_Toc136851843)

[1.2 Реализация с использованием LDD 9](#_Toc136851844)

[1.3 Реализация с использованием специальных утилит и библиотек 10](#_Toc136851845)

[1.4 Обоснование выбранных методов и алгоритмов 13](#_Toc136851846)

[2 СТРУКТУРНОЕ ПРОЕКТИРОВАНИЕ 14](#_Toc136851847)

[2.1 driver.h 14](#_Toc136851848)

[2.2 driver.c 14](#_Toc136851849)

[3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 16](#_Toc136851850)

[3.1 Функция NTSTATUS DriverEntry() 16](#_Toc136851851)

[3.2 Функция VOID DriverUnload() 16](#_Toc136851852)

[3.3 Функция NTSTATUS MyAttachDevice() 17](#_Toc136851853)

[3.4 Функция NTSTATUS ReadComplete() 18](#_Toc136851854)

[4 ИНСТРУКЦИЯ ПОЛЬЗОВАТЕЛЯ 19](#_Toc136851855)

[ЗАКЛЮЧЕНИЕ 21](#_Toc136851856)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 22](#_Toc136851857)

[ПРИЛОЖЕНИЕ А (обязательное) Листинг кода программного средства 23](#_Toc136851858)

[ПРИЛОЖЕНИЕ Б (обязательное) Скриншоты работы программы 34](#_Toc136851859)

[ПРИЛОЖЕНИЕ В (обязательное) Блок-схемы алгоритмов функций DispatchRead() и DispatchPass() 36](#_Toc136851860)

[ПРИЛОЖЕНИЕ Г (обязательное) Ведомость документов 39](#_Toc136851861)

# ВВЕДЕНИЕ

На сегодняшний день драйверы устройств являются важными компонентами программного обеспечения, которые играют решающую роль в обеспечении взаимодействия между операционной системой и аппаратным обеспечением компьютера. Драйверы обычно пишутся производителем устройств и распространяются вместе с устройством. Поскольку для каждой операционной системы требуются специальные драйверы, производители устройств обычно поставляют драйверы для нескольких наиболее популярных операционных систем.

В рамках проекта необходимо разработать драйвер устройства. Устройством, на которое ориентирован драйвер, была выбрана клавиатура. Операционная система, для которой реализован проект, — Windows 10. Сам проект представляет собой драйвер фильтра клавиатуры, который считывает нажатия клавиш и через отладочные сообщения выводит информацию о скан-коде клавиш и их состоянии(нажатие/отпускание), при нажатии на клавишу Caps Lock блокирует определенные клавиши и выводит отладочные сообщения о блокировке, аналогичным образом разблокирует клавиши.

В ранние годы компьютерной истории, когда компьютеры были простыми и состояли из небольшого числа компонентов, драйверы не были отдельно выделенными компонентами. Компьютеры того времени были специализированными системами, и устройства были прямо связаны с компьютером без посредников. С появлением более сложных компьютерных систем и развитием операционных систем, стало ясно, что необходимо разработать программное обеспечение, которое бы обеспечивало взаимодействие между операционной системой и устройствами.

Один из первых примеров драйверов устройств можно найти в операционной системе UNIX, которая появилась в 1969 году. В UNIX драйверы представляли собой набор программ, обеспечивающих взаимодействие между ядром операционной системы и аппаратным обеспечением. Они были написаны специально для конкретных устройств и предоставляли абстрактный интерфейс для взаимодействия с ними.

С развитием компьютерной индустрии и появлением новых устройств, таких как принтеры, сканеры, модемы и другие периферийные устройства, стало необходимо стандартизировать драйверы для обеспечения совместимости с различными операционными системами и устройствами. В 1980-х годах были разработаны стандарты, такие как стандарт IEEE 1275 для управления периферийными устройствами в компьютерах.

Цель курсового проектирования заключается в использовании полученных знаний, умений и навыков в ходе изучения дисциплины «Операционные системы и системное программирование» с последующей разработкой программного обеспечения, разработанного для организации связи между операционной системой компьютера и физическими устройствами.

Для достижения цели были поставлены следующие задачи: изучить необходимую информацию по алгоритму и разработать программный продукт.

Курсовое проектирование представляет собой учебно-исследовательскую деятельность и необходимо определить объект и предмет исследования. Объектом исследования является программа «драйвер устройства». Предметом исследования является изучение фреймворка WDK для Windows 10 с последующей разработкой драйвера.

# 1 ОБЗОР ЛИТЕРАТУРЫ

Драйвер устройства - это программное обеспечение, разработанное для обеспечения связи между операционной системой и конкретным устройством. Он служит посредником, переводя команды и запросы операционной системы в понятный язык для конкретного устройства и обеспечивая его правильную работу.

Роль драйвера устройства:

* Обеспечение совместимости: Драйверы устройств играют важную роль в обеспечении совместимости между операционной системой и устройством. Каждое устройство имеет свой собственный набор команд и протоколов, и драйверы переводят эти команды в понятный язык операционной системы. Без драйвера устройство не сможет корректно функционировать и взаимодействовать с операционной системой.
* Управление ресурсами: Драйверы устройств контролируют доступ к ресурсам устройства, таким как память, процессорное время и порты ввода-вывода. Они эффективно распределяют ресурсы между различными приложениями и процессами, использующими устройство. Например, если несколько приложений одновременно пытаются обратиться к принтеру, драйвер управляет очередностью печати и распределяет ресурсы таким образом, чтобы каждое приложение получило доступ к устройству в нужный момент.
* Предоставление дополнительных функций: Некоторые драйверы устройств предоставляют дополнительные функции и возможности, которые не доступны в операционной системе. Например, драйверы графических карт могут предлагать дополнительные настройки и эффекты для улучшения графического опыта пользователя. Драйверы звуковых карт могут предоставлять расширенные аудиофункции, такие как эффекты обработки звука или поддержка многоканального звука.
* Обеспечение безопасности: Драйверы устройств играют важную роль в обеспечении безопасности системы. Они могут предотвратить несанкционированный доступ к устройствам и помочь защитить систему от вредоносных программ. Некоторые драйверы могут включать функции контроля доступа, шифрования данных или проверки подлинности, чтобы обеспечить безопасность и конфиденциальность информации.
* Обновление и поддержка: Драйверы устройств также нуждаются в обновлениях и поддержке со стороны производителя. Постоянное обновление драйверов позволяет исправлять ошибки, улучшать производительность и добавлять новые функции. Производители обычно предоставляют обновления драйверов через свои официальные веб-сайты или с помощью программного обеспечения для управления драйверами.

## Реализация с использованием WDK

WDK (Windows Driver Kit) - это пакет разработчика драйверов для операционных систем семейства Windows, предоставляемый компанией Microsoft. WDK включает в себя набор инструментов, документацию и примеры кода, необходимые для разработки драйверов устройств под операционные системы Windows.

Основные компоненты WDK включают:

* Инструменты разработки: WDK предоставляет различные инструменты разработки, такие как компиляторы, отладчики, линкеры и другие средства, которые помогают разработчикам создавать и отлаживать драйверы устройств. Они включают в себя интегрированную среду разработки (IDE) Visual Studio, а также набор командной строки для выполнения различных задач разработки драйверов.
* Драйверы устройств: WDK предоставляет драйверы устройств, которые являются примерами кода для разработчиков. Эти драйверы могут служить основой для создания собственных драйверов и предоставляют примеры реализации основных функций и возможностей драйверов.
* Документация: WDK включает в себя обширную документацию, которая охватывает различные аспекты разработки драйверов устройств под Windows. Она содержит руководства, справочные материалы, API-документацию и другие ресурсы, которые помогают разработчикам понять основы и продвинутые аспекты разработки драйверов.
* Тестовые инструменты: WDK предоставляет инструменты для тестирования драйверов устройств, включая фреймворки и утилиты для проведения функциональных, нагрузочных и совместимостных тестов. Эти инструменты помогают разработчикам убедиться в правильной работе и совместимости драйверов на различных конфигурациях и операционных системах Windows.

WDK позволяет разработчикам создавать драйверы устройств для различных типов устройств, включая звуковые карты, видеокарты, сетевые адаптеры, принтеры, сканеры и многое другое. Он обеспечивает необходимые инструменты и ресурсы для упрощения и ускорения разработки драйверов, а также для обеспечения их правильной работы и совместимости с операционными системами Windows.

WDK поддерживает различные версии операционных систем Windows, включая Windows 10, Windows 8.1, Windows 8, Windows 7 и Windows Server. Кроме того, WDK обеспечивает поддержку разных архитектур процессоров, таких как x86, x64 и ARM.

С использованием WDK разработчики могут создавать драйверы устройств, которые могут обеспечивать доступ к функциональным возможностям и ресурсам устройств, оптимизировать производительность и стабильность работы системы, а также обеспечивать совместимость с другими компонентами операционной системы и приложениями.

Кроме того, WDK обеспечивает поддержку различных типов драйверов, включая драйверы устройств Plug and Play, драйверы устройств с поддержкой энергосбережения, драйверы файловых систем, драйверы сетевых устройств и другие.

## Реализация с использованием LDD

Linux Device Driver (LDD) — это фреймворк и набор инструментов для разработки драйверов устройств под операционную систему Linux. Он предоставляет разработчикам средства и ресурсы для создания драйверов, которые обеспечивают взаимодействие между аппаратными устройствами и ядром Linux.

LDD включает в себя следующие основные компоненты и функциональность:

* + Базовые концепции драйверов устройств: LDD предоставляет обзор и понимание основных концепций, связанных с разработкой драйверов устройств под Linux. Это включает понимание модели устройств в Linux, взаимодействие с ядром операционной системы, методы регистрации и идентификации устройств, работу с файловой системой и другие ключевые аспекты.
  + API и интерфейсы: LDD предоставляет набор API, которые разработчики могут использовать при создании драйверов. Это включает интерфейсы для работы с устройствами ввода/вывода, управления памятью, работой с прерываниями, взаимодействием с файловой системой и другими функциональными возможностями ядра Linux.
  + Примеры кода: LDD предоставляет обширную коллекцию примеров кода, которые помогают разработчикам лучше понять и освоить различные аспекты разработки драйверов устройств. Примеры кода демонстрируют основные шаблоны и приемы, используемые в разработке драйверов, и позволяют быстро начать создание собственных драйверов.
  + Инструменты разработки: LDD включает в себя различные инструменты, которые помогают разработчикам в процессе создания и отладки драйверов. Это может включать отладчики, статические анализаторы, профилировщики и другие инструменты, которые упрощают разработку и оптимизацию драйверов.

LDD является важным компонентом разработки драйверов устройств под Linux и предоставляет необходимые ресурсы и средства для создания эффективных и надежных драйверов. Он упрощает процесс разработки, предоставляя разработчикам знания, инструменты и примеры кода, а также обеспечивает соответствие требованиям и стандартам Linux для драйверов устройств. Благодаря LDD разработчики могут создавать драйверы, которые хорошо интегрируются с ядром Linux, обеспечивают эффективную работу устройств и взаимодействуют с другими компонентами операционной системы.

Одним из преимуществ LDD является его открытость и поддержка сообществом. Благодаря активному сообществу разработчиков Linux и открытому исходному коду, LDD постоянно развивается и обновляется, учитывая последние изменения и новые возможности операционной системы.

Кроме того, LDD обеспечивает поддержку различных типов устройств, включая устройства ввода/вывода, сетевые устройства, звуковые карты, графические устройства и другие. Это позволяет разработчикам создавать драйверы для широкого спектра устройств, обеспечивая их правильную работу и совместимость с Linux.

## 1.3 Реализация с использованием специальных утилит и библиотек

В процессе разработки драйверов могут использоваться различные утилиты и библиотеки для облегчения работы.

Вот примеры некоторых из них:

1. Libusb — это кроссплатформенная библиотека, которая предоставляет простой и удобный интерфейс для работы с устройствами USB. Она поддерживает различные операционные системы, включая Linux, Windows, macOS, FreeBSD и другие.

С помощью libusb разработчики могут осуществлять следующие операции:

* Обнаружение устройств USB: libusb позволяет сканировать систему и обнаруживать подключенные устройства USB. Разработчики могут получить информацию о дескрипторах устройств, идентификаторах производителя и продукта, интерфейсах и других характеристиках.
* Установка и удаление драйверов устройств: libusb позволяет управлять драйверами устройств, устанавливая или удаляя их в системе. Это полезно, когда требуется создание виртуальных устройств или перенастройка драйверов.
* Чтение и запись данных: libusb обеспечивает возможность чтения и записи данных на устройства USB. Разработчики могут передавать данные в формате байтов или структур данных через соответствующие точки входа или выхода устройства.
* Управление управляющими сообщениями: libusb поддерживает отправку и прием управляющих сообщений, которые позволяют взаимодействовать с устройствами USB и изменять их конфигурацию.

1. Libpcap — это библиотека, которая предоставляет возможности захвата и анализа сетевых пакетов. Она широко используется при разработке драйверов сетевых устройств, сетевых анализаторов, мониторинга сети и других сетевых приложений.

Основной функционал libpcap включает:

* Захват пакетов: libpcap позволяет открыть сетевой интерфейс и захватить пакеты, проходящие через этот интерфейс. Разработчики могут определить фильтры захвата, чтобы выбирать только интересующие их пакеты.
* Анализ пакетов: libpcap предоставляет функции для анализа содержимого захваченных пакетов. Разработчики могут извлекать информацию, такую как адреса источника и назначения, протоколы, заголовки и тела пакетов.
* Запись и воспроизведение пакетов: libpcap позволяет записывать захваченные пакеты в файлы и воспроизводить их позже для дальнейшего анализа или отладки.
* Мониторинг сети: libpcap может использоваться для создания инструментов мониторинга сети, которые анализируют и отображают активность сетевых пакетов в реальном времени.

1. ALSA — это аудиофреймворк в Linux, предоставляющий API и инструменты для работы с звуковыми картами и другими аудиоустройствами. Он является стандартом звуковой подсистемы в Linux и предлагает множество функций для управления аудиофункциональностью.

Основные возможности ALSA включают:

* Управление аудиоустройствами: ALSA позволяет открывать, закрывать и управлять аудиоустройствами, такими как звуковые карты и виртуальные звуковые устройства. Разработчики могут выполнять операции, такие как чтение и запись аудиоданных, установка параметров звука, управление громкостью и т. д.
* Многоканальная обработка: ALSA поддерживает многоканальное аудио, что позволяет работать с несколькими аудиопотоками одновременно. Разработчики могут создавать драйверы и приложения, обеспечивающие поддержку многоканального звука.
* Обработка аудиоданных: ALSA предоставляет API для обработки аудиоданных, включая преобразование формата данных, смешивание нескольких аудиопотоков, управление эффектами и другие операции обработки аудио.

1. WinAPI - это набор функций и интерфейсов, предоставляемых операционной системой Windows. Он предлагает разработчикам широкий набор инструментов и функций для создания драйверов устройств и других приложений под Windows.

WinAPI обеспечивает доступ к различным функциям и ресурсам операционной системы, включая:

* Работу с файлами и директориями: WinAPI предоставляет функции для создания, открытия, чтения и записи файлов, управления атрибутами файлов и директорий, управления путями и другие операции файловой системы.
* Работу с реестром: WinAPI позволяет работать с реестром Windows, который используется для хранения настроек и конфигурации системы. Разработчики могут использовать WinAPI для чтения, записи и изменения записей в реестре.
* Работу с сетью: WinAPI предоставляет функции для создания и управления сетевыми соединениями, обмена данными по сети, настройки сетевых интерфейсов и другие операции сетевого взаимодействия.
* Создание и управление окнами: WinAPI позволяет создавать и управлять окнами приложений, обрабатывать сообщения оконного процесса, реагировать на пользовательский ввод и другие операции, связанные с интерфейсом пользователя.

## 1.4 Обоснование выбранных методов и алгоритмов

Для выполнения поставленной задачи был выбран метод реализации с помощью WDK (Windows Driver Kit).

WDK (Windows Driver Kit) — это набор инструментов, документации и примеров кода, предоставляемый Microsoft для разработки драйверов устройств под операционные системы Windows.

Windows является наиболее популярной операционной системой для простого пользователя ПК, поэтому во время решения вопроса выбора платформы, Windows оказался лучшим вариантом.

WDK помогает разработчикам создавать надежные и совместимые драйверы, обеспечивая доступ к функциональным возможностям устройств и оптимальную работу системы. WDK является важным инструментом для разработчиков, которые стремятся создавать качественные драйверы устройств под платформу Windows.

# 2 СТРУКТУРНОЕ ПРОЕКТИРОВАНИЕ

Код программы разбит на два файла — Driver.h и Driver.cpp.

## 2.1 driver.h

В этом файле объявлены функции драйвера:

* + NTSTATUS DriverEntry(PDRIVER\_OBJECT DriverObject, PUNICODE\_STRING RegistryPath)
  + NTSTATUS MyAttachDevice(PDRIVER\_OBJECT DriverObject)
  + NTSTATUS DispatchRead(PDEVICE\_OBJECT DeviceObject, PIRP Irp)
  + NTSTATUS ReadComplete(PDEVICE\_OBJECT DeviceObject, PIRP Irp, PVOID Context),
  + NTSTATUS DispatchPass(PDEVICE\_OBJECT DeviceObject, PIRP Irp)
  + VOID DriverUnload(PDRIVER\_OBJECT DriverObject)

## 2.2 driver.c

В файле driver.c описаны функции драйвера:

* Функция VOID DriverUnload(PDRIVER\_OBJECT DriverObject) — отключает устройство, ожидает завершения операций с ним, удаляет устройство и выводит отладочное сообщение о выгрузке драйвера.
* Функция NTSTATUS DispatchPass(PDEVICE\_OBJECT DeviceObject, PIRP Irp) — передает IRP (пакет запроса ввода-вывода) следующему драйверу в стеке.
* Функция NTSTATUS ReadComplete(PDEVICE\_OBJECT DeviceObject, PIRP Irp, PVOID Context) — обработка завершения операции чтения. Она проверяет, нажата ли клавиша Caps Lock, блокирует определенные клавиши, выводит отладочные сообщения и уменьшает счетчик ожидающих операций.
* Функция NTSTATUS DispatchRead(PDEVICE\_OBJECT DeviceObject, PIRP Irp) — перенаправляет IRP на чтение следующему драйверу в стеке, устанавливает функцию обратного вызова для обработки завершения операции чтения и увеличивает счетчик ожидающих операций.
* Функция NTSTATUS MyAttachDevice(PDRIVER\_OBJECT DriverObject) — создает устройство, присоединяет его к цепочке устройств, устанавливает соответствующие флаги и возвращает статус выполнения.
* Функция NTSTATUS DriverEntry(PDRIVER\_OBJECT DriverObject, PUNICODE\_STRING RegistryPath) — вход в драйвер. Она устанавливает функцию выгрузки драйвера, определяет функции обработки различных типов IRP и вызывает функцию MyAttachDevice для создания и присоединения устройства.

# 3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Для лучшего понимания кода и работы программы ниже приведен пошаговый алгоритм работы функций NTSTATUS DriverEntry(), NTSTATUS MyAttachDevice(), NTSTATUS ReadComplete(), VOID DriverUnload();. Блок-схемы функций NTSTATUS DispatchRead() и NTSTATUS DispatchPass() приведены в приложении В(В1, В2 соответственно)

## 3.1 Функция NTSTATUS DriverEntry()

1. Объявить переменную status типа NTSTATUS;
2. Объявить переменную i типа int и инициализировать ее значением 0;
3. Установить функцию выгрузки драйвера DriverUnload в DriverObject->DriverUnload;
4. Начать цикл с условием i = 0, i < IRP\_MJ\_MAXIMUM\_FUNCTION, i++, при выполнении условия перейти к шагу 5, иначе к шагу 6;
5. Установить функцию обработки DispatchPass в DriverObject->MajorFunction[i], перейти к шагу 4;
6. Установить функцию обработки DispatchRead для IRP\_MJ\_READ в DriverObject->MajorFunction[IRP\_MJ\_READ];
7. Вызвать функцию MyAttachDevice с передачей DriverObject в качестве аргумента и сохранить результат в status;
8. Если статус выполнения не является успешным, вывести отладочное сообщение "Attaching is FAILING." и перейти к шагу 10;
9. Иначе, вывести отладочное сообщение "Attaching is SUCCESS." И перейти к шагу 10;
10. Вернуть статус выполнения;
11. Конец функции.

## 3.2 Функция VOID DriverUnload()

1. Инициализировать переменную interval типа LARGE\_INTEGER со значением 0;
2. Получить указатель на DeviceObject из DriverObject;
3. Задать значение interval.QuadPart равным -10 \* 1000 \* 1000;
4. Отсоединить устройство с помощью функции IoDetachDevice и передать LowerKbdDevice из расширения устройства DeviceObject;
5. Войти в цикл с условием пока pendingkey не равно нулю, перейти к шагу 6, если условие не выполняется, перейти к шагу 7;
6. Задержать выполнение потока с помощью функции KeDelayExecutionThread и передать режим KernelMode, флаг FALSE и адрес переменной interval, перейти к шагу 5;
7. Удалить устройство с помощью функции IoDeleteDevice и передать myKbdDevice;
8. Вывести отладочное сообщение "Driver is UNLOAD.";
9. Конец функции.

## 3.3 Функция NTSTATUS MyAttachDevice()

1. Объявить переменную status типа NTSTATUS;
2. Создать переменную TargetDevice типа UNICODE\_STRING и инициализировать ее значением "\\Device\\KeyboardClass0";
3. Создать устройство с помощью функции IoCreateDevice, передав DriverObject, размер расширения устройства sizeof(DEVICE\_EXTENSION), NULL, тип файла FILE\_DEVICE\_KEYBOARD, флаг 0, флаг FALSE и адрес переменной myKbdDevice;
4. Если создание устройства не удалось (статус выполнения не является успешным), вернуть статус выполнения и перейти к шагу 10;
5. Установить флаги DO\_BUFFERED\_IO и снять флаг DO\_DEVICE\_INITIALIZING у myKbdDevice;
6. Очистить память расширения устройства myKbdDevice с помощью функции RtlZeroMemory, передав размер sizeof(DEVICE\_EXTENSION);
7. Присоединить устройство с помощью функции IoAttachDevice, передав myKbdDevice, &TargetDevice и адрес переменной ((PDEVICE\_EXTENSION)myKbdDevice->DeviceExtension)->LowerKbdDevice;
8. Если присоединение устройства не удалось (статус выполнения не является успешным), удалить устройство с помощью функции IoDeleteDevice и вернуть статус выполнения, перейти к шагу 10;
9. Вернуть STATUS\_SUCCESS;
10. Конец функции.

## 3.4 Функция NTSTATUS ReadComplete()

1. Создать массив строк keyflag размером 4 и заполнить его значениями "KeyDown", "KeyUp", "E0", "E1";
2. Получить указатель на массив структур Keys типа PKEYBOARD\_INPUT\_DATA из SystemBuffer в Irp;
3. Вычислить количество структур в массиве Keys и сохранить результат в переменную structnum;
4. Инициализировать переменную i равной 0;
5. Если статус выполнения Irp равен STATUS\_SUCCESS, перейти к шагу 6, инече к шагу 15;
6. Начать цикл с условиями i=0, i<structnum, i++, если условия выполняются перейти к шагу 7, иначе к шагу 15;
7. Проверить, является ли нажатой клавиша Caps Lock с помощью условия Keys[i].MakeCode == 0x3A && Keys[i].Flags == 0;
8. Если клавиша Caps Lock нажата, изменить значение переменной CapsLockPressed на противоположное;
9. Если переменная CapsLockPressed равна истине, перейти к шагу 10, иначе к шагу 14;
10. Начать цикл с условиями j=0, j<46, j++, если условия выполняются перейти к шагу 11, иначе к шагу 14;
11. Проверить, совпадает ли Keys[i].MakeCode со значением scanCodeMs[j];
12. Если значения совпадают, вывести отладочное сообщение "The button is block, scan code = %x.", иначе перейти к шагу 10;
13. Восстановить память структуры Keys[i] с помощью функции RtlZeroMemory, передав размер sizeof(KEYBOARD\_INPUT\_DATA) и перейти к шагу 10;
14. Вывести отладочное сообщение "The SCAN CODE is %x (%s)\n", передав Keys[i].MakeCode и keyflag[Keys->Flags];
15. Если у Irp установлен флаг PendingReturned, вызвать функцию IoMarkIrpPending и передать Irp;
16. Уменьшить значение pendingkey на 1;
17. Вернуть статус выполнения Irp с помощью Irp->IoStatus.Status;
18. Конец функции.

# 4 ИНСТРУКЦИЯ ПОЛЬЗОВАТЕЛЯ

Как было сказано ранее, это проект по созданию драйвера фильтра клавиатуры, который считывает нажатия клавиш и через отладочные сообщения выводит информацию о скан-коде клавиш и их состояние(нажатие/отпускание), при нажатии на клавишу Caps Lock блокирует определенные клавиши и выводит отладочные сообщения о блокировке, аналогичным образом разблокирует клавиши.

Для того чтобы загрузить драйвер в операционную систему Windows вашего компьютера и запустить его, а затем остановить и выгрузить, есть несколько вариантов, например через командную строку или специальную программу.

* Загрузка, запуск, остановка и выгрузка с помощью командной строки:

1. Необходимо открыть командную строку от имени администратора.
2. С помощью команды SC, сокращенно “Service Control”, используемой в операционной системе Windows для управления службами, прописать “SC CREATE <name> binPath= <path/to/driver.sys> type= kernel” для регистрации драйвера в системе, где <name> — имя, под которым пользователь хочет зарегистрировать драйвера в системе, <path/to/driver.sys> —полный путь к файлу драйвера.
3. Далее при успешной регистрации драйвера запустить его, прописав команду “SC START <name>”.
4. Для остановки работы драйвера прописать команду “SC STOP <name>”. Следует отметить, что после ввода этой команды будет запущено ожидание операции чтения. Это означает, что перед тем как драйвер остановит свою работу, пользователь должен нажать любую клавишу на клавиатуре. Это необходимо, чтобы убедиться, что все операции чтения завершены перед остановкой работы драйвера.
5. Для удаления драйвера из системы, прописать команду “SC DELETE <name>”.

* Загрузка, запуск, остановка и выгрузка с помощью программы KmdManager[1]:

1. Запустить KmdManager от имени администратора.
2. Внутри программы выбрать файл драйвера.
3. Нажать кнопку “Register” для регистрации драйвера в системе, после чего будет выведено соответствующее сообщение (приложение Б1).
4. Нажать кнопку “Run” для запуска драйвера, после чего будет выведено соответствующее сообщение (приложение Б1).
5. Для остановки работы драйвера нажать кнопку “Stop”. Следует отметить, что после нажатия этой кнопки будет запущено ожидание операции чтения. Это означает, что перед тем как драйвер остановит свою работу, пользователь должен нажать любую клавишу на клавиатуре. Это необходимо, чтобы убедиться, что все операции чтения завершены перед остановкой работы драйвера (приложение Б5).
6. Для удаления драйвера из системы нажать кнопку “Unregister” (приложение Б5).

* Описание работы с драйвером, когда он успешно зарегистрирован и запущен:

1. Нажать клавиши клавиатуры, после чего драйвер обработает их и выведет отладочные сообщения со скан-кодом клавиш и их состоянием в программу DebugView[2], после чего будут выведено соответствующие сообщения (приложение Б2).
2. По желанию пользователя возможна блокировка клавиш по нажатию клавиши Caps Lock, после чего будет выведено соответствующее сообщение (приложение Б3).
3. По желанию пользователя возможна разблокировка клавиш посредством повторного нажатия клавиши Caps Lock (приложение Б4).

# ЗАКЛЮЧЕНИЕ

В этом курсовом проекте был реализован драйвер устройства на основе изученного материала и знаний из дисциплины "Операционные системы и системное программирование". Основными плюсами этой программы являются: возможность отображения скан-кодов клавиш, возможность блокировки некоторых клавиш.

В дальнейшем данный проект может быть улучшен в следующих аспектах:

1. Обработка множественных нажатий клавиш.
2. Изменение раскладки клавиатуры.
3. Обработка специальных клавиш и мультимедийных функций.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

[1] KmdManager [Электронный ресурс]. — Режим доступа: [Driver Development Tools Pack](https://driver-development-tools-pack.software.informer.com/) 1.3 – [Дата доступа] — 10.05.2023.

[2] DebugView for Windows [Электронный ресурс]. — Режим доступа: [DebugView](https://debugview.en.softonic.com/)– [Дата доступа] — 10.05.2023.

3.  В. Комиссарова, Программирование драйверов для Windows. — СПб.: БХВ-Петербург, 2007. — 256 с.: ил. — (Профессиональное программирование) — с. 7 – 65.

4. Использование Visual Studio или MSBuild для создания драйвера [Электронный ресурс]. — Режим доступа: [Создание драйвера с помощью WDK](https://learn.microsoft.com/ru-ru/windows-hardware/drivers/develop/building-a-driver) – [Дата доступа] — 04.05.2023.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода программного средства

\\ файл “driver.h”

#include <wdf.h>

#include <ntddk.h>

NTSTATUS DriverEntry(PDRIVER\_OBJECT DriverObject, PUNICODE\_STRING RegistryPath);

NTSTATUS MyAttachDevice(PDRIVER\_OBJECT DriverObject);

NTSTATUS DispatchRead(PDEVICE\_OBJECT DeviceObject, PIRP Irp);

NTSTATUS ReadComplete(PDEVICE\_OBJECT DeviceObject, PIRP Irp, PVOID Context);

NTSTATUS DispatchPass(PDEVICE\_OBJECT DeviceObject, PIRP Irp);

VOID DriverUnload(PDRIVER\_OBJECT DriverObject);

\\ файл “driver.с”

#include "driver.h"

USHORT scanCodeMs[85] = {

0x01,

0x47,

0x30,

0x31,

0x32,

0x33,

0x34,

0x35,

0x29,

0x02,

0x03,

0x04,

0x05,

0x06,

0x07,

0x08,

0x09,

0x0A,

0x0B,

0x0C,

0x0D,

0x10,

0x11,

0x12,

0x13,

0x14,

0x15,

0x16,

0x17,

0x18,

0x19,

0x20,

0x21,

0x22,

0x23,

0x24,

0x25,

0x26,

0x27,

0x28,

0x1A,

0x1B,

0x2B,

0x1E,

0x1F,

0x2C,

0x2D,

0x2E,

0x2F,

0x48,

0x49,

0x4B,

0x4C,

0x4D,

0x4F,

0x50,

0x51,

0x00,

0x37,

0x4A,

0x4E,

0x52,

0x53,

0x1C,

0x39,

0x36,

0x0E,

0x0F,

0x2A,

0x1D,

0x38,

0x3F,

0x3C,

0x3D,

0x3E,

0x40,

0x41,

0x42,

0x43,

0x44,

0x57,

0x58,

0x49,

0x51,

0x1D

};

typedef struct

{

PDEVICE\_OBJECT LowerKbdDevice;

}DEVICE\_EXTENSION, \* PDEVICE\_EXTENSION;

typedef struct \_KEYBOARD\_INPUT\_DATA

{

USHORT UnitId;

USHORT MakeCode;

USHORT Flags;

USHORT Reserved;

ULONG ExtraInformation;

} KEYBOARD\_INPUT\_DATA, \* PKEYBOARD\_INPUT\_DATA;

PDEVICE\_OBJECT myKbdDevice = NULL;

ULONG pendingkey = 0;

BOOLEAN CapsLockPressed = FALSE;

VOID DriverUnload(PDRIVER\_OBJECT DriverObject)

{

LARGE\_INTEGER interval = { 0 };

PDEVICE\_OBJECT DeviceObject = DriverObject->DeviceObject;

interval.QuadPart = -10 \* 1000 \* 1000;

IoDetachDevice(((PDEVICE\_EXTENSION)DeviceObject->DeviceExtension)->LowerKbdDevice);

while (pendingkey)

{

KeDelayExecutionThread(KernelMode, FALSE, &interval);

}

IoDeleteDevice(myKbdDevice);

DbgPrint("Driver is UNLOAD.");

}

NTSTATUS DispatchPass(PDEVICE\_OBJECT DeviceObject, PIRP Irp)

{

IoCopyCurrentIrpStackLocationToNext(Irp);

return IoCallDriver(((PDEVICE\_EXTENSION)DeviceObject->DeviceExtension)->LowerKbdDevice, Irp);

}

NTSTATUS ReadComplete(PDEVICE\_OBJECT DeviceObject, PIRP Irp, PVOID Context)

{

CHAR\* keyflag[4] = { "KeyDown", "KeyUp", "E0", "E1" };

PKEYBOARD\_INPUT\_DATA Keys = (PKEYBOARD\_INPUT\_DATA)Irp->AssociatedIrp.SystemBuffer;

int structnum = Irp->IoStatus.Information / sizeof(KEYBOARD\_INPUT\_DATA);

int i;

if (Irp->IoStatus.Status == STATUS\_SUCCESS)

{

for (i = 0; i < structnum; i++)

{

// Проверить, нажата ли клавиша Caps Lock

if (Keys[i].MakeCode == 0x3A && Keys[i].Flags == 0)

{

CapsLockPressed = !CapsLockPressed;

}

if (CapsLockPressed)

{

for (int j = 0; j < 85; j++)

{

if (Keys[i].MakeCode == scanCodeMs[j])

{

// Заблокировать клавишу

DbgPrint("The button is block, scan code = %x.", scanCodeMs[j]);

RtlZeroMemory(&Keys[i], sizeof(KEYBOARD\_INPUT\_DATA));

}

}

}

DbgPrint("The SCAN CODE is %x (%s)\n", Keys[i].MakeCode, keyflag[Keys->Flags]);

}

}

if (Irp->PendingReturned)

{

IoMarkIrpPending(Irp);

}

pendingkey--;

return Irp->IoStatus.Status;

}

NTSTATUS DispatchRead(PDEVICE\_OBJECT DeviceObject, PIRP Irp)

{

IoCopyCurrentIrpStackLocationToNext(Irp);

IoSetCompletionRoutine(Irp, ReadComplete, NULL, TRUE, TRUE, TRUE);

pendingkey++;

return IoCallDriver(((PDEVICE\_EXTENSION)DeviceObject->DeviceExtension)->LowerKbdDevice, Irp);

}

NTSTATUS MyAttachDevice(PDRIVER\_OBJECT DriverObject)

{

NTSTATUS status;

UNICODE\_STRING TargetDevice = RTL\_CONSTANT\_STRING(L"\\Device\\KeyboardClass0");

status = IoCreateDevice(DriverObject, sizeof(DEVICE\_EXTENSION), NULL, FILE\_DEVICE\_KEYBOARD, 0, FALSE, &myKbdDevice);

if (!NT\_SUCCESS(status))

{

return status;

}

myKbdDevice->Flags |= DO\_BUFFERED\_IO;

myKbdDevice->Flags &= ~DO\_DEVICE\_INITIALIZING;

RtlZeroMemory(myKbdDevice->DeviceExtension, sizeof(DEVICE\_EXTENSION));

status = IoAttachDevice(myKbdDevice, &TargetDevice, &((PDEVICE\_EXTENSION)myKbdDevice->DeviceExtension)->LowerKbdDevice);

if (!NT\_SUCCESS(status))

{

IoDeleteDevice(myKbdDevice);

return status;

}

return STATUS\_SUCCESS;

}

NTSTATUS DriverEntry(PDRIVER\_OBJECT DriverObject, PUNICODE\_STRING RegistryPath)

{

NTSTATUS status;

int i;

DriverObject->DriverUnload = DriverUnload;

for (i = 0; i < IRP\_MJ\_MAXIMUM\_FUNCTION; i++)

{

DriverObject->MajorFunction[i] = DispatchPass;

}

DriverObject->MajorFunction[IRP\_MJ\_READ] = DispatchRead;

status = MyAttachDevice(DriverObject);

if (!NT\_SUCCESS(status))

{

DbgPrint("Attaching is FAILING.");

}

else

{

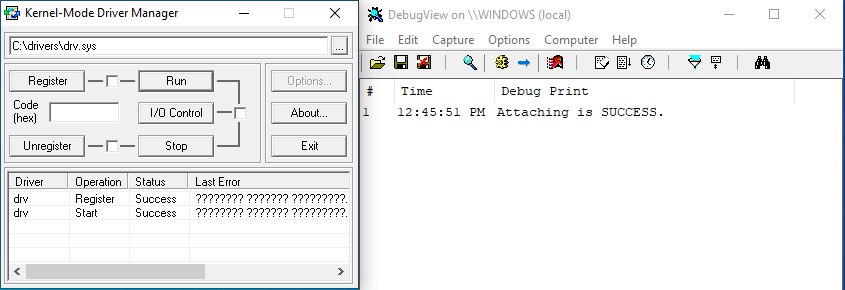
DbgPrint("Attaching is SUCCESS.");

}

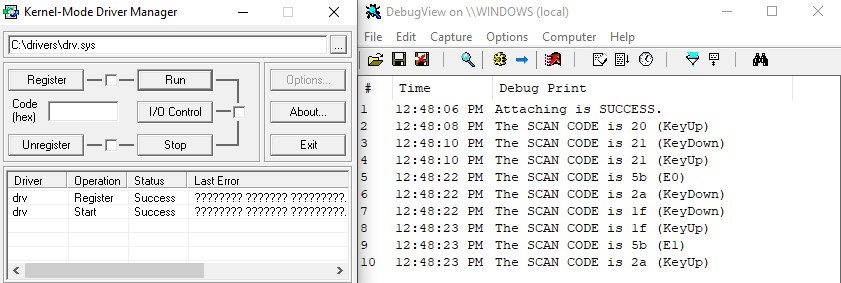
return status;

}

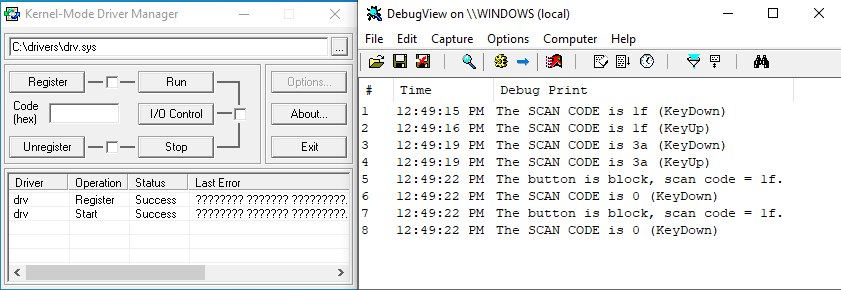
# ПРИЛОЖЕНИЕ Б (обязательное) Скриншоты работы программы

****

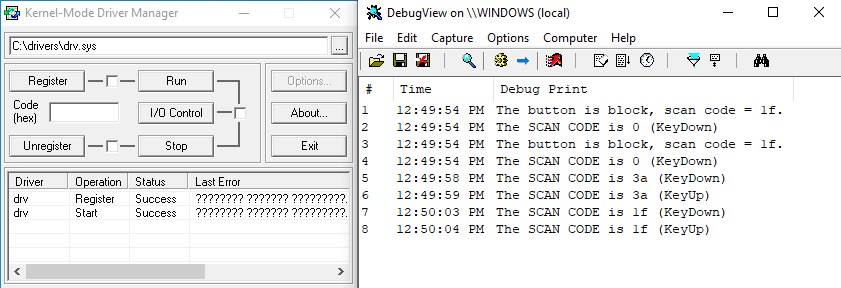
Б.1 — Загрузка драйвера



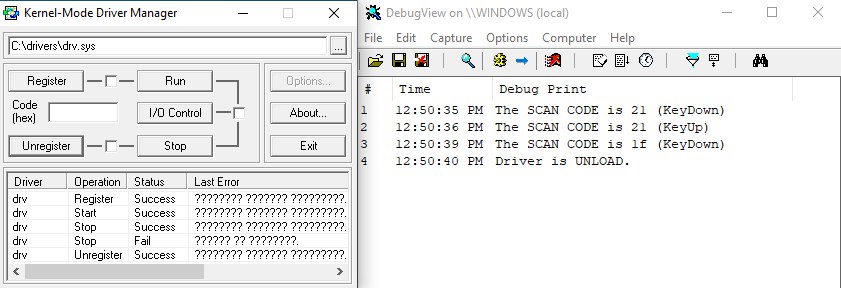
Б.2 — Вывод скан-кодов нажатых клавиш



Б.3 — Блокировка клавиш по нажатию Caps Lock

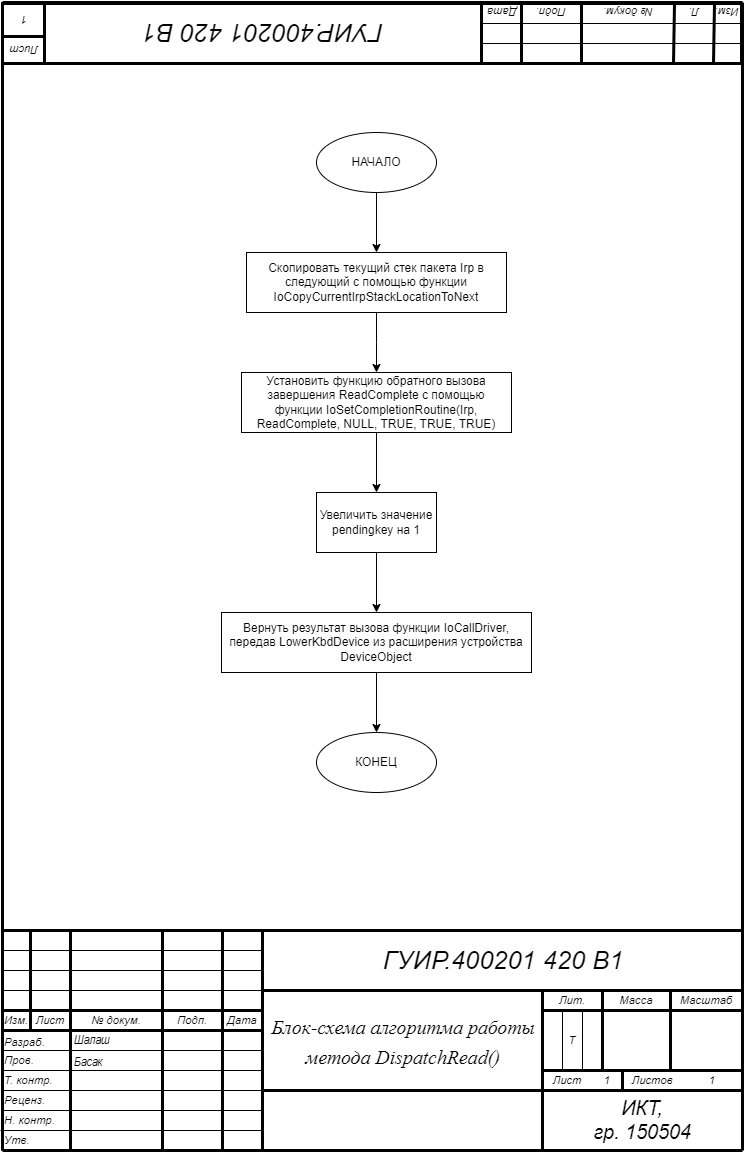


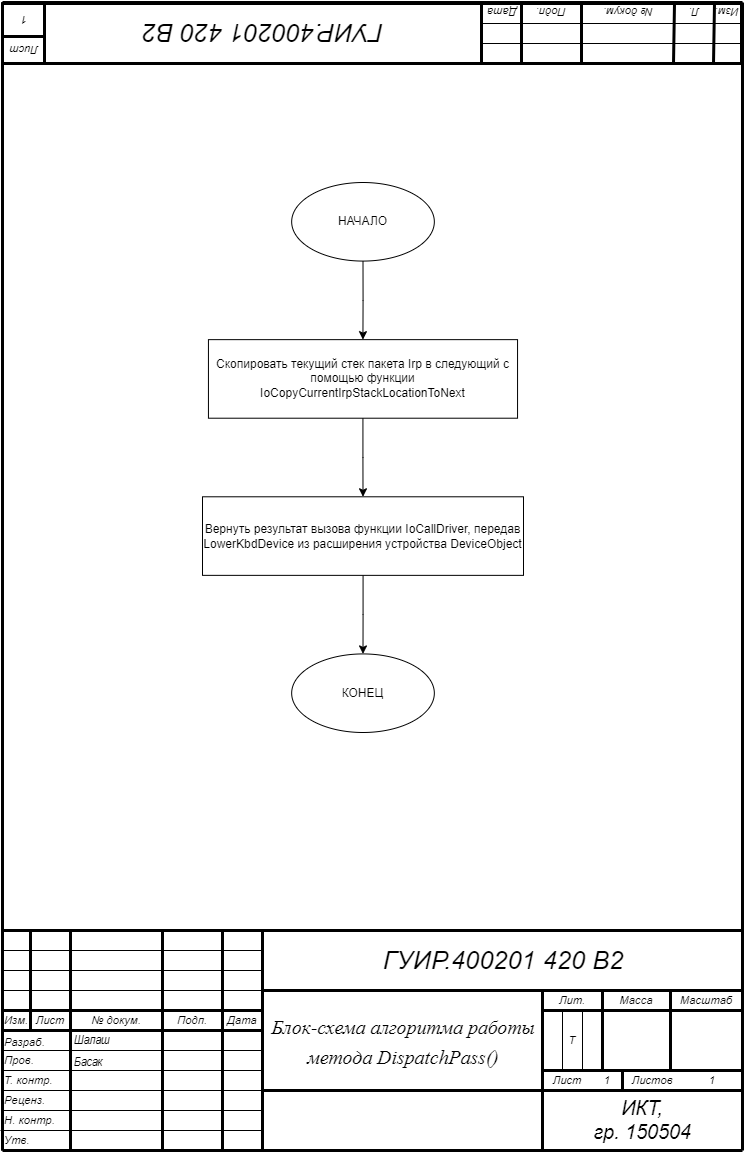
Б.4 — Разблокировка клавиш при повторном нажатии Caps Lock



Б.5 — Выгрузка драйвера

# ПРИЛОЖЕНИЕ В (обязательное) Блок-схемы алгоритмов функций DispatchRead() и DispatchPass()





# ПРИЛОЖЕНИЕ Г (обязательное) Ведомость документов

