|  |
| --- |
| **Министерство образования Республики Беларусь**  **Учреждение образования**  **«Брестский государственный технический университет»**  **Кафедра ИИТ**  **ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**  **К КУРСОВОМУ ПРОЕКТУ ПО ДИСЦИПЛИНЕ**  **«Системное программирование»**  **Тема: «****Драйвер виртуального монитора для ОС Windows»**  **КП.ПО-9.1-40-01-01**  **Листов: 23**  **Выполнил:**  Студент 3-го курса,  ФЭИС,  Группы ПО-9  Мисиюк А. С.  **Нормоконтроль:**  Козик И.Д.  **Проверил:**  Козик И.Д.  **Брест 2023** |

# ВВЕДЕНИЕ

Курсовой проект на тему "*Драйвер виртуального монитора для операционной системы Windows*" представляет собой ценный опыт в области разработки драйверов и программного обеспечения под Windows. Изучение данной темы позволяет погрузиться в глубины операционной системы, а также освоить тонкости взаимодействия с аппаратными компонентами компьютера. В рамках проекта будет рассмотрен частично процесс создания графического драйвера, начиная с анализа модели и структур графических драйверов и заканчивая тестированием и испытанием его работы на реальном стационарном компьютере. Такой опыт не только расширит понимание работы операционной системы, но и позволит освоить навыки разработки на более низком уровне абстракции, что является важным компонентом в профессиональной карьере в сфере разработки ПО.

Среда разработки – Microsoft Visual Studio, а также наборы инструментов: SDK и WDK 10.

Операционная система, под которую создается драйвер – Windows 10. Хотя драйвер создается по универсальной модели и подойдет для ОС Windows от 8-й версии.

Система контроля версий – Git. Проект, а также готовые файлы проекта (включая драйвер) и инструкцию по использованию, можно найти на публичном репозитории по адресу: *https://github.com/11ALX11/SP\_kurz*/.

Альтернативно инструкция представлена в Приложении А.

# 1. **ЧТО ТАКОЕ ДРАЙВЕР**

## 1.1. **Введение в драйвер**

Многие считают что самому создать драйвер для Windows это что-то на грани фантастики. Но на самом деле это не так. Конечно, разработка драйвера для какого-то навороченного девайса бывает не простой задачей. Но ведь тоже самое можно сказать про создание сложных программ или игр.

Сложно дать одно точное определение термина "драйвер". В самом фундаментальном смысле драйвер — это программный компонент, который позволяет операционной системе и устройству взаимодействовать друг с другом.

Например, предположим, что приложению необходимо считывать некоторые данные с устройства. Приложение вызывает функцию, реализованную операционной системой, а операционная система вызывает функцию, реализованную драйвером. Драйвер, написанный той же компанией, которая разработала и изготовила устройство, знает, как взаимодействовать с оборудованием устройства для получения данных. После того как драйвер получает данные с устройства, он возвращает данные в операционную систему, которая возвращает их приложению (рис. 1).

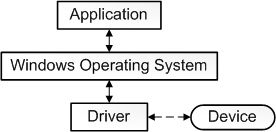


Рисунок 1 – простая схема взаимодействия драйвера и системы

**Развертывание определения.**

Наше объяснение до сих пор упрощено несколькими способами:

* **Не все драйверы должны быть написаны компанией, которая разработала устройство.**

Во многих случаях устройство разработано в соответствии с опубликованным стандартом оборудования. Таким образом, драйвер может быть написан корпорацией Майкрософт, и конструктор устройств не должен предоставлять драйвер.

* **Не все драйверы взаимодействуют напрямую с устройством.**

Для заданного запроса ввода-вывода (например, чтения данных с устройства) в стеке драйверов часто имеется несколько драйверов, участвующих в запросе. Обычный способ визуализации стека — первый участник вверху и последний участник в нижней части, как показано на этой схеме. Некоторые драйверы в стеке могут участвовать в преобразовании запроса из одного формата в другой. Эти драйверы не взаимодействуют напрямую с устройством; они просто управляют запросом и передают запрос драйверам, которые находятся ниже в стеке (рис. 2).

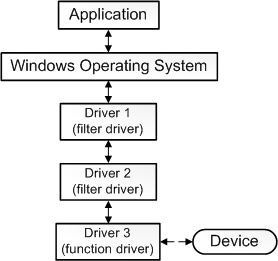


Рисунок 2 – схема взаимодействия драйверов (стека драйверов) с системой

**Драйвер функции**. Драйвер функции в стеке, который напрямую взаимодействует с устройством, называется драйвером функции.

**Драйвер фильтра**. Драйверы, выполняющие вспомогательную обработку, называются драйверами фильтров.

* Некоторые драйверы фильтров отслеживают и записывают сведения о запросах ввода-вывода, но не участвуют в них активно. Например, некоторые драйверы фильтров выступают в качестве проверяющих, чтобы убедиться, что другие драйверы в стеке обрабатывают запрос ввода-вывода правильно.

Мы можем расширить определение драйвера , сказав, что драйвер — это любой программный компонент, который наблюдает или участвует в обмене данными между операционной системой и устройством.

**Драйверы программного обеспечения**

Расширенное определение является достаточно точным, но по-прежнему неполным, так как некоторые драйверы вообще не связаны с каким-либо аппаратным устройством.

Например, предположим, что вам нужно написать средство, которое имеет доступ к основным структурам данных операционной системы. Доступ к этим структурам можно получить только с помощью кода, выполняемого в режиме ядра. Это можно сделать, разделив средство на два компонента. Первый компонент выполняется в пользовательском режиме и представляет пользовательский интерфейс. Второй компонент выполняется в режиме ядра и имеет доступ к данным основной операционной системы. Компонент, работающий в пользовательском режиме, называется приложением, а компонент, работающий в режиме ядра, называется программным драйвером. Программный драйвер не связан с аппаратным устройством.

На рисунке 3 показана схема, на которой приложение в пользовательском режиме взаимодействует с программным драйвером в режиме ядра.

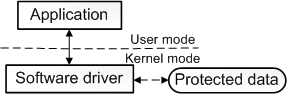


Рисунок 3 – взаимодействие пользовательского режима с режимом ядра

Программные драйверы всегда работают в режиме ядра. Главной причиной написания программного драйвера является получение доступа к защищенным данным, доступным только в режиме ядра. Однако драйверам устройств не всегда требуется доступ к данным и ресурсам в режиме ядра. Поэтому некоторые драйверы устройств работают в пользовательском режиме.

**Дополнительные сведения о драйверах функций**

Наше объяснение до сих пор упрощено определение драйвера функции. Мы сказали, что драйвер-функция для устройства — это драйвер в стеке, который напрямую взаимодействует с устройством. Это верно для устройства, которое подключается напрямую к шине PCI. Драйвер функции для устройства PCI получает адреса, сопоставленные с ресурсами порта и памяти на устройстве. Драйвер функции напрямую взаимодействует с устройством, записывая их на эти адреса.

Однако во многих случаях устройство не подключается напрямую к шине PCI. Вместо этого устройство подключается к адаптеру шины узла, который подключен к шине PCI. Например, USB-тостер подключается к адаптеру шины узла (называемому контроллером узла USB), который подключен к шине PCI. Usb-тостер имеет драйвер функции, а USB-контроллер узла также имеет драйвер функции. Драйвер функции для тостера косвенно взаимодействует с тостером, отправляя запрос драйверу функции для контроллера узла USB. Затем драйвер функции для хост-контроллера USB напрямую взаимодействует с оборудованием USB-контроллера узла, которое взаимодействует с тостером.

## 1.2. **Пользовательский режим** и **режим ядра**

Процессор на компьютере под управлением Windows имеет два разных режима: пользовательский режим и режим ядра.

Процессор переключается между двумя режимами в зависимости от типа кода, выполняемого на процессоре. Приложения выполняются в пользовательском режиме, а основные компоненты операционной системы — в режиме ядра. Хотя многие драйверы работают в режиме ядра, некоторые драйверы могут работать в пользовательском режиме.

**Пользовательский режим**

При запуске приложения в пользовательском режиме Windows создает процесс для приложения. Процесс предоставляет приложению частное виртуальное адресное пространство и таблицу частных дескрипторов. Так как виртуальное адресное пространство приложения является частным, одно приложение не может изменять данные, принадлежащие другому приложению. Каждое приложение выполняется изолированно, и в случае сбоя приложения сбой ограничивается одним приложением. Другие приложения и операционная система не затрагиваются сбоем.

Помимо частного, виртуальное адресное пространство приложения в пользовательском режиме ограничено. Процесс, выполняющийся в пользовательском режиме, не может получить доступ к виртуальным адресам, зарезервированным для операционной системы. Ограничение виртуального адресного пространства приложения в пользовательском режиме предотвращает изменение и, возможно, повреждение критически важных данных операционной системы.

**Режим ядра**

Весь код, который выполняется в режиме ядра, использует одно виртуальное адресное пространство. Таким образом, драйвер в режиме ядра не изолирован от других драйверов и самой операционной системы. Если драйвер, работающий в режиме ядра, случайно выполняет запись на неправильный виртуальный адрес, данные, принадлежащие операционной системе или другому драйверу, могут быть скомпрометированы. При сбое драйвера в режиме ядра происходит сбой всей операционной системы.

На рисунке 4 изображена схема, на которой показано взаимодействие между компонентами пользовательского режима и режима ядра.

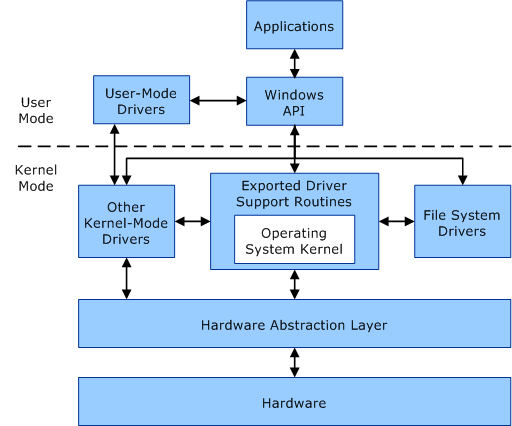


Рисунок 4 - взаимодействие между компонентами пользовательского и режима ядра

# 2. **МОДЕЛЬ WINDOWS DISPLAY DRIVER MODEL (WDDM)**

## 2.1. Архитектура WDDM

Модель драйвера дисплея Windows (WDDM) — это архитектура драйвера графического дисплея, представленная в Windows Vista (WDDM 1.0).

Для модели windows Display Driver Model (WDDM) требуется, чтобы поставщик графического оборудования предоставил сопряженный драйвер отображения пользовательского режима (UMD) и драйвер отображения в режиме ядра (KMD; также называется драйвером минипорта дисплея).

Архитектура модели драйвера дисплея WDDM состоит из частей режима пользователя и режима ядра. На рисунке 5 показана архитектура, необходимая для поддержки WDDM.

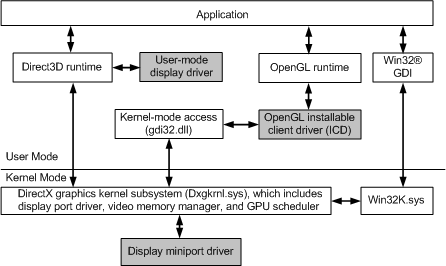


Рисунок 5 – архитектура WDDM

Поставщик графического оборудования должен предоставить драйвер дисплея в пользовательском режиме и драйвер мини-порта дисплея (также известный как драйвер отображения в режиме ядра или KMD).

* Драйвер отображения в пользовательском режиме — это библиотека динамической компоновки (DLL), загружаемая средой выполнения Direct3D;
* Драйвер мини-порта дисплея взаимодействует с подсистемой ядра графики DirectX.

## 2.2. Поток операций windows Display Driver Model (WDDM)

На следующей схеме показан поток операций WDDM, выполняемых с момента создания устройства отрисовки до отображения содержимого на дисплее. Сведения, приведенные на схеме, более подробно описывают упорядоченную последовательность потока операций.

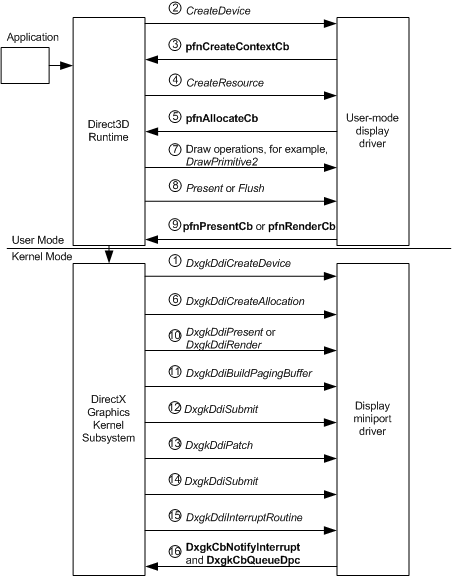


Рисунок 6 – поток операций WDDM

* **Создание устройства отрисовки**

После того, как приложение запросит создание устройства отрисовки, выполните приведенные далее действия.

1. Подсистема ядра графики DirectX (Dxgkrnl) вызывает функцию DxgkDdiCreateDevice драйвера минипорта дисплея (KMD).   
   KMD инициализирует прямой доступ к памяти (DMA), возвращая указатель на заполненную структуру DXGK\_DEVICEINFO в элементе pInfoструктуры DXGKARG\_CREATEDEVICE;
2. Если вызов DxgkDdiCreateDevice завершается успешно, среда выполнения Direct3D вызывает функцию CreateDevice драйвера отображения пользовательского режима (UMD);
3. В вызове CreateDevice UMD должен явным образом вызвать функцию pfnCreateContextCb среды выполнения для создания одного или нескольких контекстов GPU, которые являются потоками выполнения GPU на только что созданном устройстве. Среда выполнения возвращает сведения в UMD в элементах pCommandBuffer и CommandBufferSize структуры D3DDDICB\_CREATECONTEXT для инициализации буфера команд;

* **Создание поверхностей для устройства**

После того как приложение запросит создание поверхностей для устройства отрисовки, выполните следующие действия.

1. Среда выполнения Direct3D вызывает функцию CreateResource UMD;
2. CreateResource вызывает предоставленную средой выполнения функцию pfnAllocateCb;
3. Среда выполнения вызывает функцию DXGkDdiCreateAllocation KMD, указывая количество и типы создаваемых выделений. DxgkDdiCreateAllocation возвращает сведения о выделениях в массиве DXGK\_ALLOCATIONINFO структур в элементе pAllocationInfoструктуры DXGKARG\_CREATEALLOCATION;

* **Отправка буфера команд в режим ядра**

После того, как приложение запросит рисование на поверхность:

1. Среда выполнения Direct3D вызывает функцию UMD, связанную с операцией рисования, например DrawPrimitive2;
2. Среда выполнения Direct3D вызывает функцию UMD Present или Flush, чтобы вызвать отправку буфера команд в режим ядра. Примечание. UMD также отправляет буфер команд, когда буфер команд заполнен;
3. В ответ на шаг 8 UMD вызывает одну из следующих функций, предоставляемых средой выполнения:
   * Функция pfnPresentCb среды выполнения, если был вызван метод Present;
   * Функция pfnRenderCb среды выполнения, если был вызван метод Flush или буфер команд заполнен.
4. функция DxgkDdiPresent в KMD вызывается, если был вызван pfnPresentCb , либо dxgkDdiRender или DxgkDdiRenderKm , если был вызван pfnRenderCb . KMD проверяет буфер команд, записывает в буфер DMA в формате оборудования и создает список выделения, описывающий используемые поверхности;

* **Отправка буфера DMA на оборудование**

1. Dxgkrnl вызывает функцию DXGkDdiBuildPagingBuffer KMD для создания буферов DMA специального назначения, которые перемещают выделения, указанные в списке выделения, в память, доступную для GPU, и из нее. Эти специальные буферы DMA называются буферами разбиения по страницам. DxgkDdiBuildPagingBuffer не вызывается для каждого кадра;
2. Dxgkrnl вызывает функцию DXGkDdiSubmitCommand KMD для постановки буферов подкачки в единицу выполнения GPU;
3. Dxgkrnl вызывает функцию DXGkDdiPatch KMD для назначения физических адресов ресурсам в буфере DMA;
4. Dxgkrnl вызывает функцию DXGkDdiSubmitCommand KMD для постановки буфера DMA в единицу выполнения GPU. Каждый буфер DMA, отправленный в GPU, содержит идентификатор ограждения, который является числом. После того как GPU завершит обработку буфера DMA, GPU создает прерывание;
5. KMD получает уведомление о прерывании в функции DxgkDdiInterruptRoutine . KMD должен считывать из GPU идентификатор ограждения только что завершенного буфера DMA;
6. KMD должен вызывать функцию DxgkCbNotifyInterrupt , чтобы уведомить DXGK о завершении буфера DMA. KMD также должен вызывать функцию DxgkCbQueueDpc для постановки в очередь отложенного вызова процедуры (DPC).

## 2.3 Преимущества WDDM

Создавать драйверы графики и дисплея проще с помощью WDDM, а не модели XDDM в Windows 2000, так как в ней реализованы следующие улучшения. Кроме того, драйверы WDDM способствуют повышению стабильности и безопасности операционной системы. Меньше кода драйвера выполняется в режиме ядра, где он может получить доступ к системным адресным пространствам и, возможно, вызвать сбои.

* Среда выполнения Direct3D и подсистема ядра графики DirectX (Dxgkrnl) выполняют больше операций обработки отображения; то есть в среде выполнения и подсистеме находится больше кода, а не в драйверах. Эта обработка включает код, который управляет видеопамятью и планирует буферы прямого доступа к памяти (DMA) для GPU;
* Для создания Surface требуется меньше этапов в режиме ядра.  
  Для создания Surface в операционных системах до Windows Vista требовались следующие последовательные вызовы режима ядра:

1. DdCanCreateSurface;
2. DdCreateSurface;
3. D3dCreateSurfaceEx.

Для создания Surface в WDDM требуется только вызов драйвера отображения в пользовательском режиме CreateResource , который, в свою очередь, вызывает функцию pfnAllocateCb среды выполнения. Этот вызов приводит к тому, что Dxgkrnl вызывает функцию DxgkDdiCreateAllocation драйвера режима ядра;

* Вызовы, которые создают и уничтожают поверхности, а также блокируют и разблокируют ресурсы, более равномерно связаны;
* WDDM одинаково обрабатывает видеопамяти, системную память и управляемые поверхности. Операционные системы до Windows Vista обрабатывали эти компоненты тонко разными способами;
* Преобразование шейдера выполняется в пользовательском режиме части драйверов дисплея.

Этот подход устраняет следующие сложности, возникающие при выполнении преобразования шейдера в режиме ядра:

* + Аппаратные модели, которые не соответствуют абстракциям интерфейса драйвера устройства (DDI);
  + Сложная технология компилятора, используемая в переводе.

Так как обработка шейдера выполняется полностью для каждого процесса и доступ к оборудованию не требуется, обработка шейдера в режиме ядра не требуется. Таким образом, код перевода шейдера можно обрабатывать в пользовательском режиме.

Необходимо написать код try/except вокруг кода перевода в пользовательском режиме. Ошибки перевода должны привести к возврату к обработке приложения.

Фоновый перевод (т. е. код перевода, который выполняется в отдельном потоке от других потоков обработки экрана) проще писать в пользовательском режиме.

# 3. **ДРАЙВЕР КОСВЕННОГО ОТОБРАЖЕНИЯ (IDD)**

В качестве драйвера для создания виртуального монитора мною был выбран драйвер косвенного отображения. Почему? В моих глазах этот драйвер наиболее подходит для поставленной задачи несмотря на то, что традиционные драйвера WDDM также подойдут. IDD (или **Indirect Display Driver**) – драйвер пользовательского режима, что послужило определенным плюсом из-за простоты и безопасности его тестирования. Также логическое применение, так как при создании виртуального монитора не подразумевается никакого отображения, поэтому косвенное отображение очень даже подходит.

Модель драйвера непрямого отображения (IDD) предоставляет простую модель драйвера в пользовательском режиме для поддержки мониторов, которые не подключены к традиционным выводам дисплея GPU. Например, аппаратный ключ, подключенный к компьютеру через USB, к которому подключен обычный монитор (VGA, DVI, HDMI, DP и т.д.).

**Реализация IDD**

IdD — это сторонний драйвер UMDF для устройства. Он разработан с использованием функциональных возможностей, предоставляемых IddCx (Класс косвенного драйвера дисплея eXtension) для взаимодействия с графическими подсистемами Windows следующими способами:

* Создание графического адаптера, представляющего устройство непрямого отображения;
* Мониторы отчетов подключены и отключены от системы;
* Укажите описания подключенных мониторов;
* Предоставление доступных режимов отображения;
* Поддержка других функциональных возможностей дисплея, таких как аппаратный курсор мыши, гамма-связь, I2C-связь и защищенное содержимое;
* Обработка изображений рабочего стола для отображения на мониторе;

Так как IDD является драйвером UMDF, он также отвечает за реализацию всех функций UMDF , таких как обмен данными с устройствами, управление питанием, plug and play и т. д.

IdD выполняется в сеансе 0 без каких-либо компонентов, запущенных в пользовательском сеансе, поэтому любая нестабильность драйвера не повлияет на стабильность системы в целом.

На рисунке 7 представлена схема, на которой обозревается архитектура:

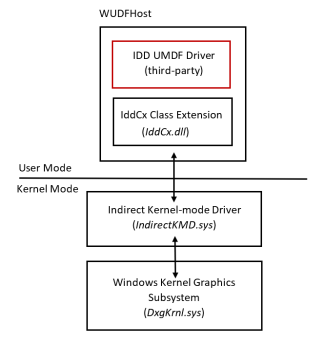


Рисунок 7 – обзор архитектуры IDD

**Модель пользовательского режима**

IdD — это модель только в пользовательском режиме без поддержки компонентов режима ядра. Таким образом, драйвер может использовать любые API DirectX для обработки образа рабочего стола. На самом деле IddCx предоставляет образ рабочего стола для кодирования в поверхности DirectX.

**Примечание.** Драйвер не должен вызывать ИНТЕРФЕЙСы API пользовательского режима, которые не подходят для использования драйверами, например GDI, ИНТЕРФЕЙСы API окон, OpenGL или Vulkan.

IdD следует создавать как универсальный драйвер Windows , чтобы его можно было использовать на нескольких платформах Windows.

Во время сборки idD UMDF объявляет версию IddCx, для которую она была создана, а ОПЕРАЦИОННая система гарантирует, что при загрузке драйвера будет загружена правильная версия IddCx.

# 4. **РЕЗУЛЬТАТЫ НАПИСАНИЯ И ИСПЫТАНИЯ IDD**

К непосредственно написанию кода драйвера можно приступать после установки необходимого ПО. Нужно установить последнюю версию Microsoft Visual Studio, наборы инструментов: SDK и WDK.

Прежде чем писать код с нуля, лучше попытаться найти образец уже готового драйвера для изучения работы структур и процессов на готовом примере, а не по сухой документации, и уже его основе реализовать требуемый функционал.

Код проекта находится в *Приложении Б*.

Состоит из 2 разделений:

* Приложение для запуска виртуального монитора временно (без установки как устройства, монитор существует до закрытия приложения);
* Непосредственно проекта самого *драйвера косвенного отображения* (*IDD*).

Дальше я укажу пару моментов, которые отличают драйвер виртуального монитора от настоящего драйвера (драйвера косвенного отображения):

* Обработка кадра (в этом проекте ее просто нет);
* Отчет по статистике кадра;
* Программно прописанный EDID и конфигурация монитора (понятное дело, это драйвер должен получать от самого монитора динамически);
* Динамическая логика контейнера вместо статического (как если бы монитор был подключен вечно, что не верно в случае с физическими мониторами, но вполне подходит в случае с виртуальным монитором).

**Испытать данный драйвер не очевидная задача.**

Даже если все будет работать правильно, ничего не измениться визуально. Так как мы создали монитор, которого “не существует”.

Достаточно провести мышку к границе справа (обычно именно сюда Windows помещает новые мониторы) и вести ее дальше. Если при попытке вернуть ее на экран она появляется не сразу, как если бы она была дальше, а не на границе, то можно подтвердить работоспособность данного драйвера виртуального монитора.

Хотя лучше зайти в **диспетчер устройств**, выбрать пункты **Видеоадаптеры** и **Мониторы**, где среди видеоадаптеров можно заметить наш адаптер, созданный драйвером, а среди мониторов появится новый монитор (как правило, их будет 2) (рис. 8).

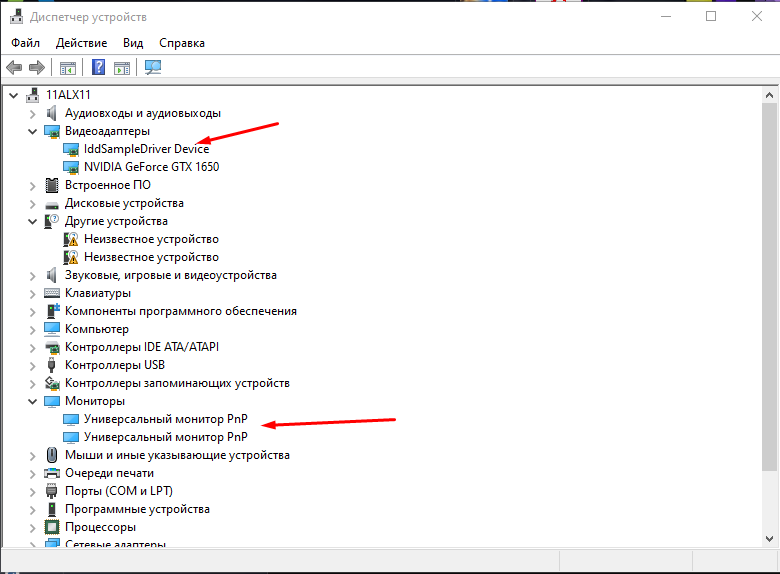


Рисунок 8 – проверка обнаружения виртуальных устройств

Также можно перейти в **Параметры** и найти **Дисплей**. Тут можно заметить, что Windows определяет 2 монитора, и при желании, 2-ому можно даже поменять разрешение на указанное программно в драйвере (рис. 9).

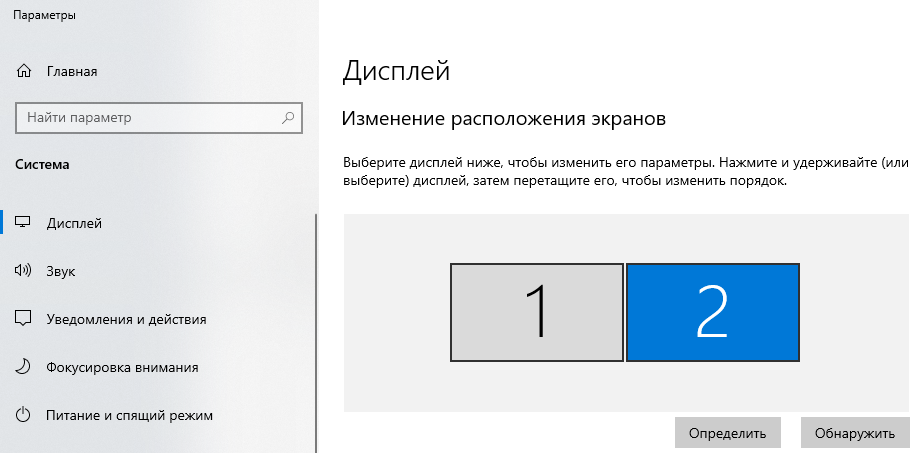


Рисунок 9 – проверка наличия 2-го дисплея

Ну и наконец, можно использовать программы захвата экрана (такие как OBS) для взаимодействия с новым экраном (рис. 10).

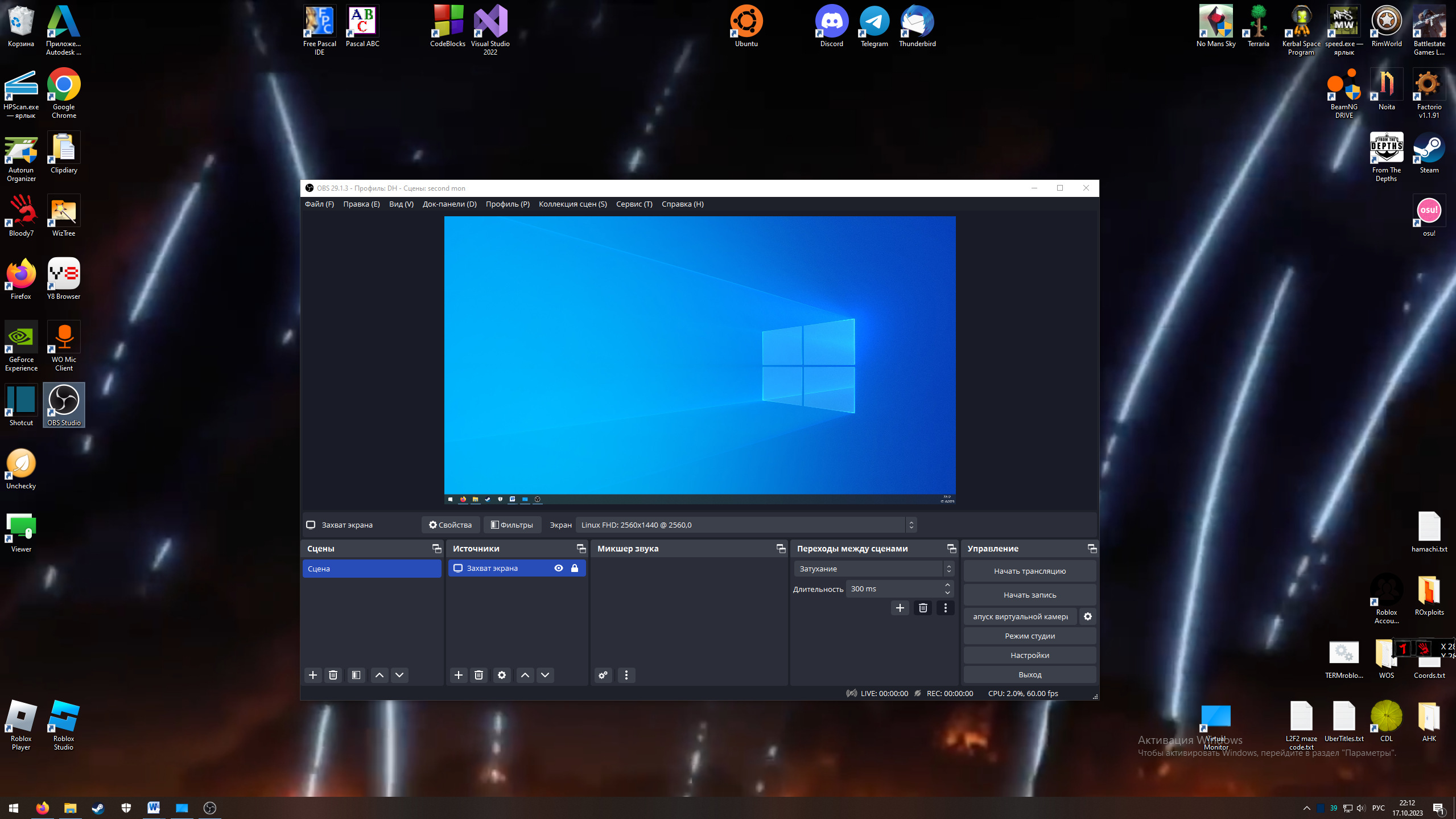


Рисунок 10 – обнаружение и проецирование 2-го дисплея программой OBS

Инструкция по использованию представлена в Приложении А.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта был разработан драйвер виртуального монитора для операционной системы Windows 10. Результаты данного проекта позволили углубить знания в области разработки драйверов и программного обеспечения, а также приобрести ценный опыт взаимодействия с аппаратными компонентами системы.

Основные достижения в рамках данного проекта включают:

1. **Разработка драйвера**: Был создан эффективный и гибкий драйвер, способный эмулировать работу дополнительного монитора в ОС Windows;
2. **Изучение операционной системы**: Работа над проектом потребовала глубокого понимания внутреннего устройства операционной системы Windows, а также принципов работы с драйверами;
3. **Навыки разработки на низком уровне**: Разработка драйвера позволила освоить навыки работы на более низком уровне абстракции, что является важным компонентом профессионального роста в области разработки ПО.

Таким образом, выполнение данного курсового проекта не только позволило успешно реализовать поставленную цель, но и обогатило набор профессиональных навыков, необходимых в сфере разработки программного обеспечения под операционную систему Windows.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

* Драйвер — это просто / Хабр:   
  https://habr.com/ru/articles/145926/ (дата доступа: 28.10.2023)
* Что такое драйвер? - Windows drivers | Microsoft Learn:   
  https://learn.microsoft.com/ru-ru/windows-hardware/drivers/gettingstarted/what-is-a-driver- (дата доступа: 28.10.2023)
* Пользовательский режим и режим ядра - Windows drivers | Microsoft Learn:   
  https://learn.microsoft.com/ru-ru/windows-hardware/drivers/gettingstarted/user-mode-and-kernel-mode (дата доступа: 28.10.2023)
* Написание первого драйвера - Windows drivers | Microsoft Learn:   
  https://learn.microsoft.com/ru-ru/windows-hardware/drivers/gettingstarted/writing-your-first-driver (дата доступа: 28.10.2023)
* Написание универсального драйвера Windows (UMDF 2) на основе шаблона - Windows drivers | Microsoft Learn:   
  https://learn.microsoft.com/ru-ru/windows-hardware/drivers/gettingstarted/writing-a-umdf-driver-based-on-a-template (дата доступа: 28.10.2023)
* Написание Hello World драйвера Windows (KMDF) - Windows drivers | Microsoft Learn:  
  https://learn.microsoft.com/ru-ru/windows-hardware/drivers/gettingstarted/writing-a-very-small-kmdf--driver (дата доступа: 28.10.2023)
* Написание универсального драйвера Windows (KMDF) на основе шаблона - Windows drivers | Microsoft Learn:   
  https://learn.microsoft.com/ru-ru/windows-hardware/drivers/gettingstarted/writing-a-kmdf-driver-based-on-a-template (дата доступа: 28.10.2023)
* Подготовка компьютера для развертывания и тестирования драйверов (WDK 10) - Windows drivers | Microsoft Learn:   
  https://learn.microsoft.com/ru-ru/windows-hardware/drivers/gettingstarted/provision-a-target-computer-wdk-8-1 (дата доступа: 28.10.2023)
* Multiple-Monitor Support in the Display Driver - Windows drivers | Microsoft Learn:   
  https://learn.microsoft.com/en-us/windows-hardware/drivers/display/multiple-monitor-support-in-the-display-driver (дата доступа: 28.10.2023)
* Driver design guides for display, graphics, and compute accelerator devices - Windows drivers | Microsoft Learn:   
  https://learn.microsoft.com/en-us/windows-hardware/drivers/display/ (дата доступа: 28.10.2023)
* Road map for the Windows Display Driver Model (WDDM) - Windows drivers | Microsoft Learn:   
  https://learn.microsoft.com/en-us/windows-hardware/drivers/display/roadmap-for-developing-drivers-for-the-windows-vista-display-driver-mo (дата доступа: 28.10.2023)
* Initializing Display Miniport and User-Mode Display Drivers - Windows drivers | Microsoft Learn:   
  https://learn.microsoft.com/en-us/windows-hardware/drivers/display/initializing-display-miniport-and-user-mode-display-drivers (дата доступа: 28.10.2023)
* Installing Test-Signed Driver Packages - Windows drivers | Microsoft Learn:   
  https://learn.microsoft.com/en-us/windows-hardware/drivers/install/installing-test-signed-driver-packages (дата доступа: 28.10.2023)
* Загрузка тестового подписанного кода - Windows drivers | Microsoft Learn:   
  https://learn.microsoft.com/ru-ru/windows-hardware/drivers/install/the-testsigning-boot-configuration-option (дата доступа: 28.10.2023)
* Indirect display driver model overview - Windows drivers | Microsoft Learn:   
  https://learn.microsoft.com/en-us/windows-hardware/drivers/display/indirect-display-driver-model-overview (дата доступа: 28.10.2023)