ПРОГРАММНЫЙ КОМПЛЕКС ДЛЯ ИСПОЛЬЗОВАНИЯ МОБИЛЬНОГО УСТРОЙСТВА В КАЧЕСТВЕ ДОПОЛНИТЕЛЬНОГО МОНИТОРА

Дадуш М.С., студент

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Ярмолик В.Н. – д-р техн. наук, профессор

В работе рассмотрены факторы, повлиявшие на идею создания программного комплекса, обоснование технических решений, а также проблемы, с которыми пришлось столкнуться во время разработки. Описаны принципы взаимодействия серверного компонента с операционной системой и передачи изображения с виртуального графического вывода на удаленное устройство.

Использование мобильного устройства в качестве дополнительного компьютерного монитора имеет ряд преимуществ перед физическими специализированными аналогами. Во-первых, финансовая сторона вопроса. С помощью специального программного обеспечения пользователь может использовать планшетный компьютер либо смартфон в роли как физического монитора, так и графического планшета, избавляясь от необходимости приобретения дорогостоящего оборудования. Во-вторых, мобильность. Использование мобильных устройств в роли компьютерного монитора избавляет от привязки к конкретному местоположению.

На сегодняшний день представлено множество программных средств для дублирования основного экрана и передачи изображения на удаленное устройство. Однако количество решений для расширения рабочего стола за счет экрана мобильного устройства минимально, при этом доступность их использования ограничивается операционными системами Windows и MacOS. Разработка подобного программного комплекса с поддержкой операционных систем семейства GNU/Linux является перспективной и позволит занять нишу рынка на фоне их растущей популярности.

Отличительная характеристика программного комплекса заключается в достижении единообразия предоставляемого функционала среди операционных систем GNU/Linux, так как системы данного класса характеризуются гетерогенностью компонентного состава. На сегодняшний день существует более сотни дистрибутивов с различными системными библиотеками, стандартными программами, графическим окружением и драйверами устройств. Некоторые дистрибутивы поставляются в разных версиях и отличаются предустановленными графическим окружением и набором прикладных приложений. Современные графические оболочки характеризуются существенными различиями в функциональных возможностях и механизмах взаимодействия с системными компонентами. Единственным унифицированным уровнем различных дистрибутивов остается ядро Linux. Отсюда вытекает необходимость использования библиотек, полагающихся только на функционал ядра, и разработки собственных драйверов.

Существующие для дистрибутивов GNU/Linux аналоги полностью полагаются на возможности графического сервера, из-за чего разработчики сталкиваются с необходимостью написания значительных объемов платформозависимого кода. Так, большинство программного обеспечения данного класса поддерживает X11, но не поддерживает или ограниченно поддерживает Wayland, что особенно критично, так как разработчики популярных дистрибутивов отказываются от X11 в пользу Wayland. Использование драйвера виртуального графического адаптера – единственный на данный момент способ нивелировать различия между графическими подсистемами.

С учетом поддержки нескольких операционных систем требуется разработать драйвер виртуального графического адаптера для каждой платформы. Обобщить кодовую базу для драйверов невозможно, так как имеются существенные различия не только в API ядра каждой операционной системы, но и в парадигме взаимодействия с его компонентами. Вне зависимости от платформы драйвер выполняет следующие функции:

1) инициализация виртуального графического адаптера;

2) управление виртуальными графическими выводами;

3) управление видеорежимами для каждого графического вывода;

4) захват изображения с вывода и передача буфера процессам пользовательского пространства (серверу).

Особенность этапа инициализации виртуального графического адаптера состоит в невозможности определить путь устройства. API Windows WDF (Windows Driver Frameworks) предоставляет функцию, благодаря которой разрешается назначить известное серверу имя устройства. Для доступа по присвоенному имени нужно открыть устройство по пути с префиксом \\.\GLOBALROOT\ [1]. В API ядра Linux аналогичная возможность отсутствует, из-за чего в процессе инициализации сервер вынужден перебирать устройства cardX, где X – целое неотрицательное число, из каталога /drv/dri, чтобы определить верный графический адаптер и отличить его от физической видеокарты.

При установлении соединения с клиентом сервер обращается к драйверу адаптера для создания виртуального графического вывода посредством отправки IOCTL (Input/Output Control) кода. Чтобы перевести вывод в активное состояние, необходимо идентифицировать монитор с помощью блока данных EDID (Extended Display Identification Data) [2], предварительно отредактировав информацию о мониторе и поддерживаемые видеорежимы.

В общем случае для полной поддержки операционной системой виртуального адаптера графики нужно разработать два драйвера: пользовательского режима (в Linux вместо драйвера пользовательского режима используется библиотека API), который содержит реализацию графического API, (DirectX в Windows, Vulkan и OpenGL в Linux), и режима ядра, занимающийся управлением памяти и взаимодействием с оборудованием (см. рис. 1). Создание библиотеки графического API требует огромных временных и человеческих ресурсов, поэтому целесообразно использовать уже имеющуюся в компьютере видеокарту для отрисовки и обойтись разработкой только драйвера режима ядра, с помощью которого производить управление виртуальными выводами и запрашивать кадры из памяти видеокарты. В Windows и Linux [4] описанная возможность уже реализована, следовательно, остается только вручную управлять переключением кадров на виртуальном выводе посредством отправки событий vblank.

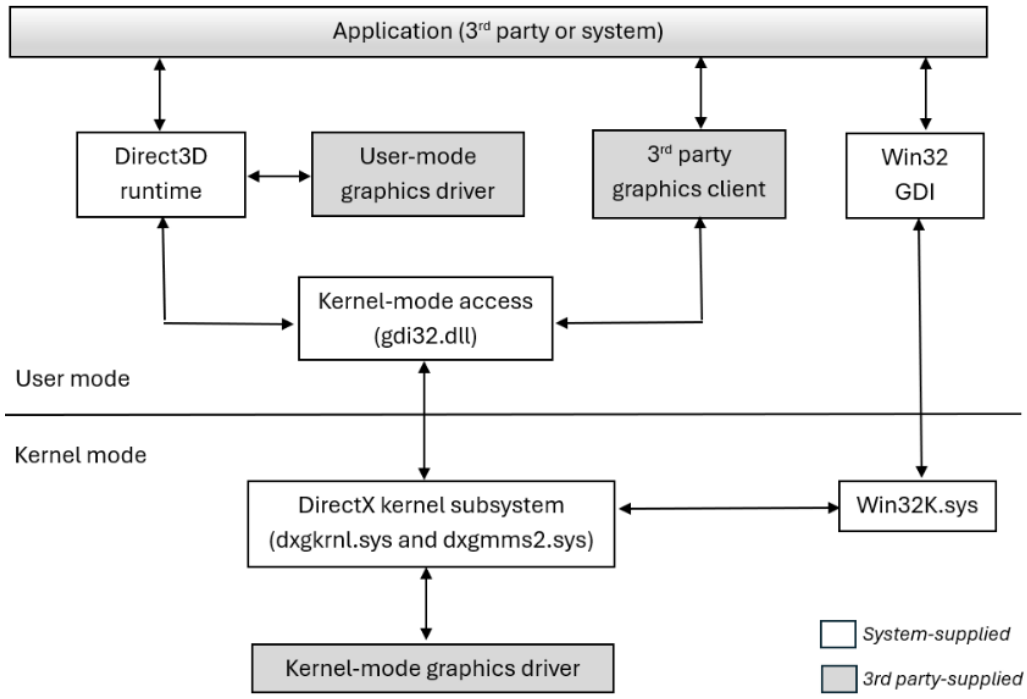

Рисунок 1 – Архитектура графического стека Windows [3]

Захват кадра осуществляется также посредством отправки IOCTL кода. Результат вызова – дескриптор буфера пикселей кадра экрана в формате ARGB. При передаче изображения по Wi-Fi критически важно использовать потоковое сжатие, так как трансляция экрана в разрешении FullHD с частотой обновления экрана 60 Гц путем отправки несжатых кадров в формате 32 бит на пиксель приводит к избыточным нагрузкам на сеть, так как средний объем передаваемых данных – 5 Гбит/с.

Таким образом, реализация графических драйверов – необходимость при реализации кроссплатформенного паритета. В отличие от существующих решений, данная разработка обеспечивает унифицированный алгоритм взаимодействия клиента и сервера: установление соединения, подключение монитора, трансляция изображения, отключение монитора – в рамках единой кодовой базы вне зависимости от используемой операционной системы и графической, подсистемы.

Список использованных источников:

1. Naming Files, Paths and Namespaces – Win32 apps : Microsoft Learn. – URL: https://learn.microsoft.com/en-us/windows/win32/fileio/naming-a-file (дата обращения: 15.03.2026).

2. VESA Enhanced EDID Standard Version 1.4 : Release A, Revision 2. – Video Electronics Standards Association, 2006. – URL: https://vesa.org/vesa-standards (дата обращения: 15.03.2026).

3. WDDM Architecture – Windows Drivers : Microsoft Learn. – URL: https://learn.microsoft.com/en-us/windows-hardware/drivers/display/windows-vista-and-later-display-driver-model-architecture (дата обращения 16.03.2026).

4. Chapter 35. PRIME Render Offload : Nvidia. – URL: https://download.nvidia.com/XFree86/Linux-x86_64/435.17/README/primerenderoffload.html (дата обращения: 16.03.2026).