Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование
информационных систем
Системное программирование

Бутрова Александра Сергеевна

Разработка графической подсистемы для встроенных систем

Выпускная квалификационная работа

Научный руководитель:
ассистент кафедры системного программирования Козлов А. П.

Рецензент:
разработчик Дерюгин Д. Е.

Санкт-Петербург
2017

SAINT PETERSBURG STATE UNIVERSITY

Software and Administration of Information Systems
System Engineering

Butrova Aleksandra

Development of graphics subsystem for embedded operating systems

Graduation Project

Scientific supervisor:
Assistant Anton Kozlov

Reviewer:
Software Developer Denis Deryugin

Saint Petersburg
2017

Оглавление

Введение

В последнее время почти каждый день любой человек сталкивается со встраиваемыми системами: на работе многие используют факс-аппараты, в больнице испытывают на себе системы компьютерной томографии и ультразвукового исследования, в магазине видят кассовые аппараты, на улице едва замечают системы наблюдения, в машине включают навигацию GPS, дома открывают умный холодильник, а на руке носят умные часы. Данный список не раскрывает использования встроенных систем ещё во многих областях деятельности человека, но показывает, что встроенные системы получили большое значение в жизни современного человека. В связи с этим одной из главных задач разработки встраиваемых систем является реализация интуитивно понятного интерфейса пользователя.

Так как встраиваемые системы решают задачи из огромного спектра задач человеческой деятельности, то каждое устройство или тип устройства должен иметь специализированный интерфейс, который будет отвечать нуждам определенного круга людей. Например, все кассиры должны уметь пользоваться кассовыми аппаратами, которые показывают информацию о товарах, их цену и окончательную стоимость. Вся функциональность ограничивается кнопками типа "убрать товар из корзины" или "рассчитать стоимость". Второй пример – умные часы – обычно требуют всего одну кнопку – кнопку переключения между режимами показа времени, калорий, пройденных шагов и других параметров. Итого, имеем экран с множественной информацией и несколькими действиями над ней и маленький экран с содержанием малого количества информации и всего один действием. Отсюда видно, что необходимы разные подходы к созданию интерфейсов.

Для создания таких интерфейсов нужны универсальные средства разработки, которые, в свою очередь, влекут за собой увеличение потребления других ресурсов, таких как память или энергопотребление. Например, для создания тонких и маленьких умных часов дополнительная память в размере карты памяти увеличит желаемые размеры.

Существует другой путь создания графического интерфейса для встраиваемых систем – создавать их с нуля. Этот путь значительно увеличит время и стоимость продукта, а также потребует занятость людей с определенными навыками.

Таким образом, постоянно возникает проблема: для каждого устройства необходимо создавать графическую подсистему, подходящую под специфичные характеристики и функциональность, выбирая один из двух путей, который потребует меньших затрат.

Одним из возможных решений является создание конфигурируемой графической подсистемы для каждого класса устройств, где конфигурируемость обозначает возможность описания устройства и указания параметров подсистемы для него с целью запуска одного и того же графического приложения на различных устройствах выбранного класса.

1. Постановка задачи

Целью квалификационной работы является разработка графической подсистемы для встроенных систем, конфигурируемой для каждого устройства вывода. Для этого необходимо выполнить задачи, описанные ниже.

1. Исследовать существующие графические подсистемы и их компоненты.
2. Разработать архитектуру конфигурируемой графической подсистемы.
3. Реализовать построенную архитектуру.
4. Произвести апробацию реализации графической подсистемы на аппаратуре.

Так как главное целью данной работы является графическая подсистема, то опишем необходимые требования, которым она должна удовлетворять:

- низкие требования к платформе(CPU, память и тд.);
- независимость от характеристик устройств вывода;
- поддержка работы с произвольным устройством графического вывода, предоставляющим кадровый буфер;
- предоставление приложению весь экран.

2. Обзор

2.1. Основные понятия

Графическая подсистема – набор функциональности, который позволяет производить общение между аппаратурой ввода-вывода и пользователем посредством графического представления информации.

Два самых известных принципов представления информации – это WIMP и POST-WIMP, которые реализуют оконную систему. WIMP – это парадигма, использующая окна, иконки, меню и курсор (windows, icons, menus, pointer). POST-WIMP, как видно из названия, – это потомок WIMP, который появился благодаря сенсорным экранам и таким технологиям как множественные касания (multitouch).

Рис. 1: Классическая схема взаимодействия основных компонентов графической подсистемы.

Такие способы представления информации предполагают использование устройства ввода (мышь, джойстик, клавиатура, тачпад, сенсорный экран и др.) для осуществления управляющих действий над окнами, меню и иконками.

На рисунке 1 представлена классическая схема графической подсистемы, которая состоит из следующих компонентов: клиент, графический интерфейс, графический сервер, оконный менеджер, ядро и аппаратура. Рассмотрим некоторые из них подробнее.

- Клиент.

  Обычно клиентами называют приложения, которые требуют отобразить свои данные на устройствах вывода.
- Графический интерфейс (graphical user interface, GUI).

  Главной задачей графического интерфейса является взаимодействие с пользователем на уровне визуализированной информации. Он использует визуальные примитивы для представления информации в удобном виде для пользователя.

  Примеры графического интерфейса: GNOME Shell, Unity, RDE Plasma, Xfce.
- Графический сервер (display server).

  Это программное обеспечение, чья главная задача состоит в том, чтобы координировать ввод и вывод информации клиентов в остальных частях операционной системы и аппаратного обеспечения, а также помогать взаимодействовать друг с другом. Графический сервер взаимодействует с пользователем по протоколу графического сервера(display server protocol) и коммуникационного протокола.
- Оконный менеджер (windowing manager)

  Это программное обеспечение, которое контролирует расположение и внешний вид окна в оконной системе. Он облегчает взаимодействие между окнами, приложениями и оконной системы.

Введем еще несколько важных понятий:

Кадровый буфер или фреймбуфер (framebuffer) [14] – это часть памяти, куда помещаются обработанные графические данные для последующей отправки их на устройство для отрисовки. Данные в буфере должны соответствовать некоторому формату, заданному устройством.

Рендеринг – процесс получения требуемого формата графических данных для последующей отрисовки на устройстве.

2.2. Существующие графические подсистемы

Одним из способов описания графических подсистем является архитектура. Приведем некоторые из них.

X Window System

X Window System [18] – клиент-серверная графическая подсистема, которая для взаимодействия клиента и устройства задействует все компоненты классической схемы графической подсистемы. Данная подсистем используется уже более 20 лет и является универсальной.

Ниже представлен алгоритм взаимодействия компонентов графической подсистемы X Window System.

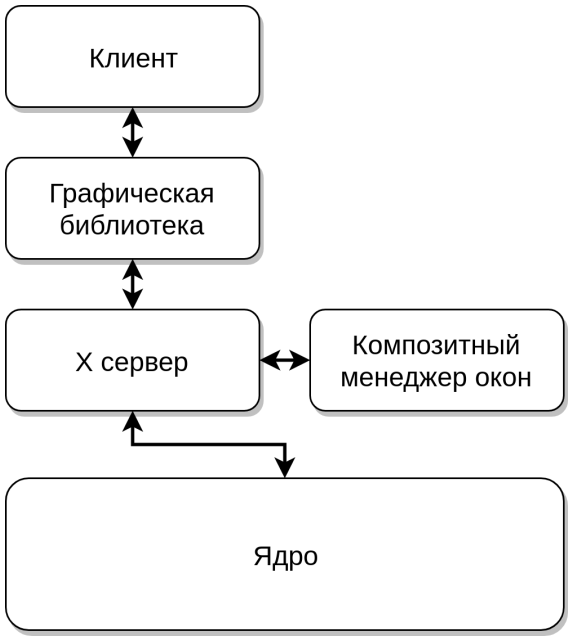

Рис. 2: Архитектура графической подсистемы X Window System.

1. Ядро получает событие от устройства ввода/вывода и передает его X серверу.
2. X сервер определяет какое окно было изменено и отправляет его к соответствующему клиенту.
3. Клиент реагирует на действия ввода и посылает рендеринг запрос обратно на сервер.
4. Сервер считает регион выделенной памяти и посылает этот участок оконному менеджеру как "поврежденный" (dirty, [12]).
5. Оконный менеджер рендерит "поврежденный" участок окна и отправляет ответ на сервер.
6. Сервер получает ответ от оконного менеджера и обновляет буфер.

Отметим несколько свойств присущих данной данной архитектуре:

- прямой доступ к фреймбуферу осуществляется через подключение дополнительных расширений;
- есть сетевая прозрачность;
- независимость от устройств ввода-вывода;
- универсальность;
- популярность;
- поддерживается множество графических библиотек;
- поддерживается использование OpenGL;
- в настоящее время продолжает развиваться и поддерживаться;
- управляет аппаратурой напрямую без использования драйверов;
- повторяет функционал из ядра.

В случае встроенных систем архитектура X Window System и ее множество возможностей (например, удаленный клиент, оконный менеджер, т.д.) являются избыточными и требуют огромных затрат ресурсов.

Wayland

Wayland [12] – это клиент-серверная графическая подсистема, в которой клиенты запрашивают отображение буфера пикселей на экране, а сервер предоставляет услуги управления отображением этих буферов.

В отличии от X Window System, Wayland позволяет клиенту и менеджеру окон общаться напрямую. Данное свойство достигнуто путем исключения сервера из архитектуры. Убрав лишнюю прослойку, архитектура Wayland не унаследовала от X Window System

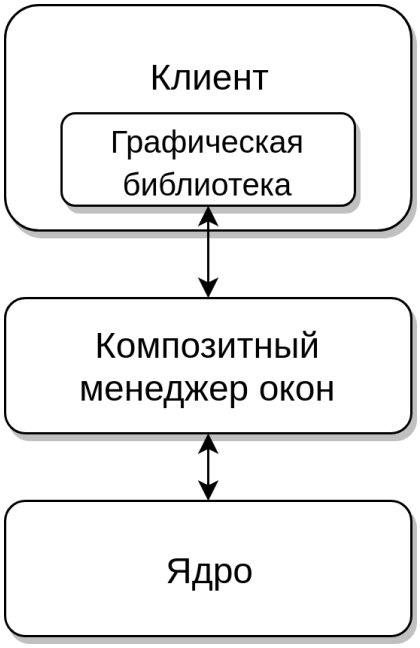

Рис. 3: Архитектура графической подсистемы Wayland.

некоторых свойств, таких как сетевая прозрачность, но, тем не менее, для использования на локальных компьютерах является оптимальным вариантом. Ниже представлена схема взаимодействия компонентов Wayland.

1. Ядро получает событие и отправляет его менеджеру окон.
2. Менеджер окон определяет какому окну принадлежит событие и преобразовывает экранные координаты в оконно-локальные координаты.
3. Клиент получает событие и обновляет интерфейс в ответ. Рендеринг происходит в клиенте, для этого клиент просто посылает запрос менеджеру окон, чтобы указать область, которая была обновлена.
4. Менеджер окон собирает события обновлений от клиентов и делает перекомпозицию (recomposite) экрана. Далее менеджер окон может выдать ioctl для планирования обновления (PageFlip).

Рассмотрим некоторые свойства Wayland:

- прямой доступ к фреймбуферу зависит от используемого композитного менеджера. Например, в эталонной реализации менеджера Weston такая функция поддерживается;
- спецификация Wayland не предусматривает сетевой прозрачности, но конкретный композитный менеджер может реализовывать это функцию;
- является универсальной системой;
- набирает популярность и имеет активно развивающееся сообщество разработчиков;
- не зависит от графических библиотек, таким образом, можем выбирать любую;
- поддерживает использование OpenGL.

Wayland является более подходящей под параметры системой, чем X Window System, но тем не менее имеет свои недостатки для случая встроенных систем, так как всё равно требует достаточно много ресурсов памяти, которые недоступны на маленьких встраиваемых системах.

Nano-X

Nano-X [5] является "уменьшенной" версией X Window System, которая разрабатывалась для встраиваемых систем. Ее архитектура состоит из трех уровней: драйвера устройств, визуализатор и API.

На уровне драйверов устройств определены интерфейсы драйверов устройств, которые используются визуализатором для выполнения аппаратных операции. Данный уровень позволяет добавлять устройства, не влияя на работу всей системы.

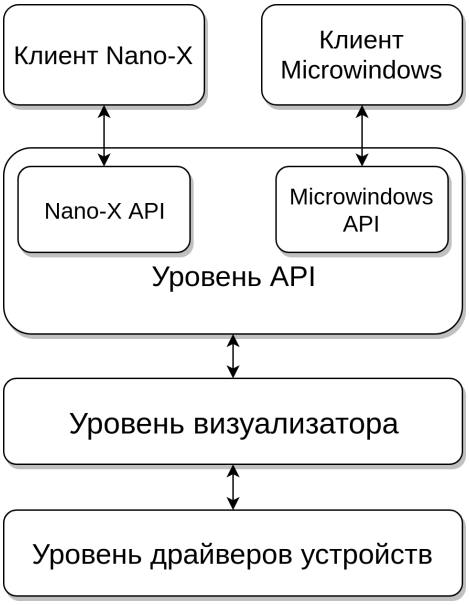

Рис. 4: Архитектура графической подсистемы Nano-X.

Уровень визуализатора не зависит от аппаратуры и является ядром функциональности Nano-X – он содержит основные операции отрисовки (отрисовка примитивов, отслеживание положения мыши, управление палитрами). Клиенты никогда не работают с этим модулем напрямую, а пользуются им через API Nano-X.

Уровень API отвечает за обработку активностей клиента и сервера, работу оконного менеджера, запросов на вывод графики и т.д. Главной задачей является инициализации устройств ввода/вывода и ожидание событий в цикле. Далее может использовать расширенные операции рисования, такие как нарисовать картинку, получить размеры и цвет окна. Определены два типа интерфейса: Microwindows API и Nano-X API. Они предоставляют совместимость с Win32 и X Window System, необходимую для облегченного портирования программ с других систем.

Некоторые свойства Nano-X:

1. поддерживается как клиент-серверная архитектура (Nano-X API),

так и сообщения-ориентированная (Microwindows API);

2. менеджер окон как часть уровня API;
3. нет новостей от официального сообщества с 2010 года;
4. поддерживается ограниченный набор графических библиотек – FLTK, NXLIB, SDK;
5. не поддерживается OpenGL.

DirectFB

DirectFB [13] – библиотека, которая работает поверх кадрового буфера и которая предоставляет слой для создания графического интерфейса. Она предоставляет такие возможности как аппаратное ускорение графики, управление устройствами ввода и абстракции интегрированной оконной системы с поддержкой множественных слоев отображения и прозрачности окон. По умолчанию DirectFB управляет только одним окном и рисует такие примитивы как треугольник, линия, прямоугольник и т.д. Для расширения возможностей используются следующие проекты:

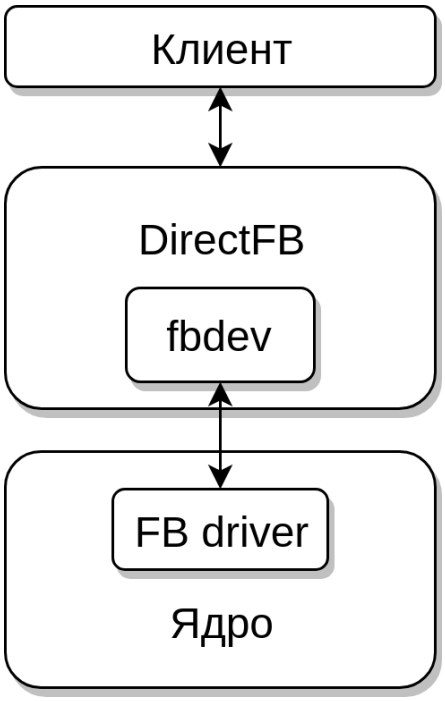

Рис. 5: Архитектура графической подсистемы DirectFB.

- SaWMan – оконный менеджер.
- ilixi – композитный менеджер.
- FusionSound – звуковая подсистема.
- FusionDale – набор вспомогательных сервисов.
- Voodoo – уровень сетевой прозрачности.
- XDirectFB – сервер для работы X Window Sysytem поверх DirectFB.

Свойства архитектуры DirectFB:

- использует прямой доступ к кадровому буферу;

- рендеринг возлагается на DirectFB, а не на клиента;
- поддерживает использование OpenGL;
- поддерживает использование популярных графических библиотек как Qt, а также специальных графических библиотек для встроенных систем (ilixi);
- существует сообщество, которое поддерживает и развивает данный проект;
- нет сетевой прозрачности;
- нет оконного менеджера.

Сравнение архитектур

В таблице 1 представлены результаты сравнения описанных выше архитектур.

Свойство\система	X Window System	Wayland	Nano-x	DirectFB
Клиент-серверная	Да	Да	Поддерживается	Нет
Рендер	Сервер	Клиент	Уровень визуализатора	DirectFB Core
Менеджер окон	Есть	Есть	Поддерживается	Нет
Прямой доступ к фреймбуферу	Дополнительно	Дополнительно	Есть	Есть
Сетевая прозрачность	Есть	Дополнительно	Нет	Нет
Популярность	Да	Да	Нет	Нет
Графические библиотеки	Много	Много	FLTK, NXLIB, SDK	Qt, ilixi, LiTE
Развитие и сообщество	Есть	Активное	Нет	Есть

Таблица 1: Результаты сравнения архитектур графических подсистем X Window System, Wayland, Nano-X, DirectFB.

2.3. Компоненты графических подсистем

Рассмотрим некоторые важные составляющие графической подсистемы.

API интерфейсы визуализации

Для низкоуровневого программирования очень важным является взаимодействие программного обеспечения с аппаратурой. Для облег-

чения этого процесса производители оборудования реализуют функции аппаратуры согласно некоторым спецификациям API. Ниже представлены некоторые преимущества использования спецификаций программного интерфейса.

- Улучшенная производительность. Так как производители оборудования сами пишут реализацию спецификаций, они делают реализуют функции оптимальным образом.
- Абстракция. Разработчики графических приложении не задумываются о низкоуровневой реализации графики, а полагаются на интерфейс.
- Единый интерфейс и кроссплатформенность. Программы, написанные с использованием спецификаций, будут работать на любой аппаратуре, поддерживающей эту спецификацию.

Существует много API итерфейсов визуализации [16]. Рассмотрим самые известные из них: OpenGL, Vulkan, Glide и Mantle.

OpenGL – это набор спецификаций для плотформонезависимого программного интерфейса 2D и 3D графики.

Обычно OpenGL используется с другими расширяющими её библиотеками, такими как GLU, GLUT, SDL, VTK, GLM. Также поверх OpenGL могут быть построены библиотеки для конкретных оконных менеджеров или дополнительная функциональность для определенного оборудования.

OpenGL – наиболее популярная и универсальная спецификации API, она хорошо подходит для написания игр и масштабных графических систем. Чтобы удобно писать графические системы с помощью OpenGL необходимо иметь достаточное большое количество памяти для всех требуемых библиотек-расширений OpenGL, что не всегда возможно на встроенных системах.

Vulkan является потомком OpenGL, а также в некотором роде совместимым с OpenGL, но имеет более высокую производительность и низкие накладные расходы благодаря использованию более низкого языка программирования при реализации.

Glide является спецификацией для трехмерной графики для процессоров Voodoo Graphics. Был популярен в 90-е благодаря тому, что занимал достаточно мало памяти. Добились такого результата путем включения в спецификацию только необходимых функции из OpenGL.

Существование проекта Glide было недолгим, но успело внедрится в создание игр. Таким образом, до сих пор существуют эмуляторы Glide для игр 90-х.

Спецификация Mantle разрабатывалась как альтернатива OpenGL для процессоров AMD. Является спецификацией более низкого уровня, чем обеспечивает более высокую производительность и улучшенное управление ресурсами.

Несмотря на все преимущества использования API интерфейсов визуализации, для встаиваемых систем очень важно количество используемой памяти, которую использование таких интерфейсов будет тратить впустую, так как можно быть уверенным, что в данный момент работает лишь с одним устройством, а при использывании другого, можно переконфигурировать систему.

Также стоит отметить, что не все производители устройств внедряют поддержку API интерфейсов визуализации в устройство, особенно, если это устройство со специфическими характеристиками и предназначением.

Итого, использование API интерфейсов визуализации в данной работе скорее принесет лишь повышенное потребление памяти, что недопустимо.

Графические библиотеки

Построением графического интерфейса занимаются графические библиотеки (graphics library, widget toolkit, GUI frameworks). Поскольку они также обладают своими особенностями, то выбор инструментария крайне важен и должен исходить из требований к задаче. Были проанализированы некоторые источники со списками существующих графических библиотек [1], [15], [17]. Первоначальный отбор библиотек для

рассмотрения производился по следующим критериям:

- язык реализации C/C++;
- кроссплатформенные;
- минимальные требования к ресурсам.

Подробнее были рассмотрены графические библиотеки NanoVG [9], TIGR [8], FLTK [4], nuklear [6] и ImGui [2].

Все графические библиотеки делятся на два типа [7]:

- Retained mode. Библиотеки сохраняют полную модель объектов, подлежащих визуализации. Свойства: проще в использовании, оптимизация визуализации.
- Immediate mode. Клиенты вызывают процесс визуализации напрямую. Свойства: лучший контроль и гибкость, потребление меньших ресурсов памяти.

Некоторые библиотеки комбинирует два типа для лучшей производительности.

Приведем таблицу сравнения характеристик выбранных графических библиотек.

Свойство\Библиотека	NanoVG	TiGR	Nuklear	ImGui	FLTK
Язык реализации	C	C	C	C++	C++
Тип	Retained	Retained	Immediate	Immediate	Retained
Рендер	OpenGL	OpenGL	Нет рендера по умолчанию	Нет рендера по умолчанию	OpenGL
Сторонние зависимости	Есть	Есть	Нет	Нет	Нет
Лицензия	Public Domain	Public Domain	Public Domain	MIT License	GNU Library Public License
Документация	Плохая	Плохая	Средняя	Средняя	Хорошая
Последний релиз	2016	2017	2017	2017	2016

Таблица 2: Таблица сравнения графических библиотек NanoVg, FLTK, Dear ImGui, Nuklear, TIGR.

Исходя из этой таблицы выбор был сделан в пользу графической библиотеки nuklear. Главным ее преимуществом является ее тип – immediate. Он позволяет потреблять меньше памяти, что очень важно во

встраиваемых системах. От ImGui выбранная библиотека nuklear отличается модульностью и языком реализации. Модульность библиотеки nuklear позволяет не включать неиспользуемые возможности библиотеки (например, не включать использование отрисовки с помощью вершинных буферов). Выбор в пользу языка программирования C был сделан в связи с тем, что он имеет более продвинутые возможности управления памятью.

3. Архитектура конфигурируемой графической подсистемы

Обычно во встраиваемых системах мы можем быть уверены, что систему использует всего один клиент или даже используется всего одно окно. В этих случаях нагрузка на графическую подсистему падает, например, нет необходимости выяснять какому окну и клиенту принадлежит событие. Таким образом, отпадает необходимость использования оконного менеджера и графического сервера, что убирает не только лишнюю прослойку, но и уменьшает потребление системных ресурсов.

Поскольку, как отмечено выше, для встроенных систем характерно вырождение функциональности графического сервера, то в них можно использовать другую архитектуру, назовем ее "непосредственной". В данной архитектуре необходимая серверная часть переносится на клиента и, таким образом, сильно упрощается система взаимодействия и уменьшается количество требуемых ресурсов. В итоге, получается, что клиенту достаточно писать напрямую (непосредственно) в видео память.

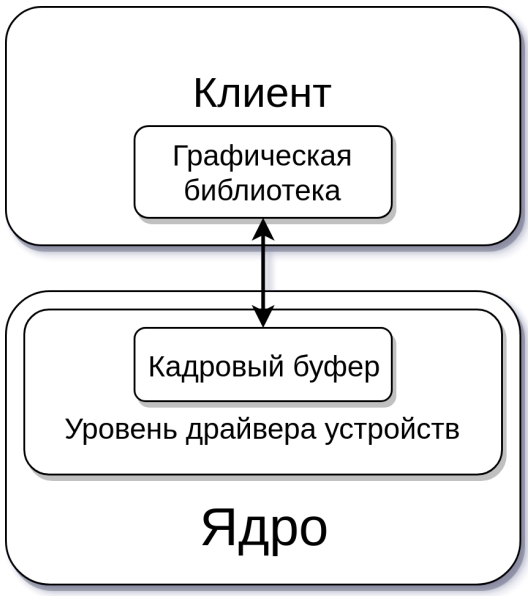

Рис. 6: Архитектура конфигурируемой графической подсистемы.

На рисунке 6 представлены основные компоненты и взаимодействие между ними. Клиент (графическое приложение) пользуется графиче-

ской библиотекой, которая умеет писать непосредственно в кадровый буфер. Чтобы добиться независимости от устройств, в модель было введено дополнительное звено – уровень драйвера устройств – которое будет использовать знания об устройстве вывода и помогать клиенту создавать графику для конкретного устройства.

Предложенная архитектура перекладывает возможность сетевого разделения графического клиента и сервера на клиента, в таком случае, позволяя клиенту потреблять необходимое количество ресурсов для вывода графической информации. Данное решение является разумным компромиссом между гибкостью и накладными ресурсами, но тем не менее идеально подходит для случаев, не требующих сетевой доступ и работу сразу нескольких приложений, которые выводят данные на экран.

Таким образом, если стоит задача минимизировать ресурсы (например, для использования в автономном "носимом" устройстве), то наиболее подходящим будет вариант с "непосредственной" архитектурой и легковесным инструментарием, а если нужно создать богатый пользовательский интерфейс, то можно выбрать другую графическую библиотеку, жертвуя ресурсами.

4. Особенности реализации

На рисунке 7 изображена архитектура конфигурируемой графической подсистемы, где в качестве компонентов взяты графическая библиотека nuklear и встраиваемая операционная система Embox.

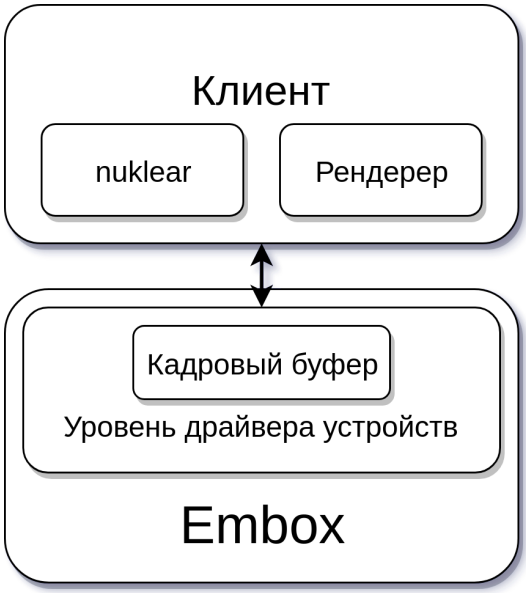

Рис. 7: Архитектура конфигурируемой графической подсистемы.

4.1. Операционная система

Графическая подсистема была реализована для встраиваемой операционной системы Embox [3]. Отметим главные её особенности:

- реализована на языке C;
- имеет свойство конфигурируемости;
- есть возможность задания параметров для каждой подсистемы.

Уровень драйвера устройств реализовывается с помощью конфигурируемости и возможности задания параметров для подсистем: разработчик может создавать модули для различных устройств ввода/вывода и статически указывать параметры для каждого из них или разрабатывать уникальный графические интерфейс, подходящий для специфических нужд.

4.2. Графическая библиотека

Графическая библиотека nuklear была выбрана из списка рассматриваемых библиотек в разделе 2.3 всвязи с обладанием следующих характеристик:

1. вся реализация находится в одном заголовочном файле;
2. имеет маленький размер библиотеки – 787,2 Кб;
3. имеет модульную структуру;
4. реализована на языке C;
5. имеет тип Immediate Mode;
6. не имеет сторонних зависимостей;
7. содержит все основные возможности рисования;
8. выпускается под лицензией Public Domain;
9. имеет документацию;
10. последний релиз в 2017 году.

Так как разрабатываемая графическая подсистема строится для класса носимых устройств, то необходимо особенно эффективно использовать память. Эффективности помогают достичь первые шесть свойств библиотеки, описанные выше.

Nuklear имеет несколько рендереров, но не имеет возможности писать непосредственно в видеопамять. Данная проблема решается в следующем разделе.

4.3. Рендерер

Для того чтобы графическая библиотека nuklear имела возможность писать непосредственно в кадровый буфер, был реализован дополнительный модуль – рендерер.

Рендерер был реализован на языке C, имеет размер 17.3 Кб и поддерживает следующую функциональность:

- запись напрямую в видеопамять;
- поддержка очереди входных элементов;

- отрисовка графических примитивов;
- отрисовка картинок;
- отрисовка текста;
- поддержка нескольких форматов цвета пикселя;
- выбор ориентации экрана.

Embox имеет интерфейс взаимодействия с кадровым буфером, поэтому главной задачей реализованного рендерера является трансляция очереди входных элементов (текст, картинки, примитивы) в данные, которые имеет формат фреймбуфера.

В операционной системе Embox фреймбур поддерживает только один примитив – квадрат. Это было основой отрисовки в реализованном рендерере: все примитивы графической библиотеки и загружаемые картинки отрисовывалось попиксельно с помощью квадрата.

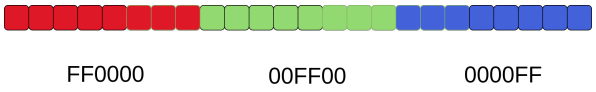

Рис. 8: 24-битный формат цвета пикселя и маски для каждого цвета.

Каждый пиксель имеет свой свой цвет в зависимости от характеристик устройств. Мной были реализованы два формата: 24-битный по 8 бит для каждого цвета (рис. 8) и 16-битный с 5, 6 и 7 битов для красного, зеленого и синего соответственно (рис. 9).

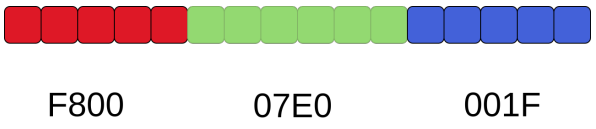

Рис. 9: 16-битный формат цвета пикселя и маски для каждого цвета.

Nuklear позволяет выбирать каким шрифтом будет выведен текст: шрифтом по умолчанию или пользовательским. Для этого необходимо завести дополнительную структуру altlas, которая хранит один или несколько цветов. В нее загружатся все желаемые шрифты. Так как Embox имеет свои шрифты по умолчанию, которые не будут занимать дополнительной памяти, реализованный рендерер отрисовывает текст только с помощью них, избегая создания дополнительной структуры.

Embox предоставляет команду для отрисовки текста. Текст может быть либо черным на белом фоне, либо белым на черном. Была добавлена возможно отрисовки текста любого цвета на любом фоне, включая прозрачный.

Также была добавлена возможно выбора ориентации экрана, которая не поддерживается в графической библиотеке.

5. Апробация

Для апробации были взяты два примера. Один из них демонстрирует возможности отрисовки примитивов, второй – возможность создания виджетов. Испытания проводились на двух платформах: виртуальной машине QEMU [10] и отладочной плате STM32F746G [11].

5.1. Апробация на виртуальной машине QEMU

Виртуальная машина QEMU имеет следующие характеристики:

- разрешение 800x600;
- формат пикселя 16 бит.

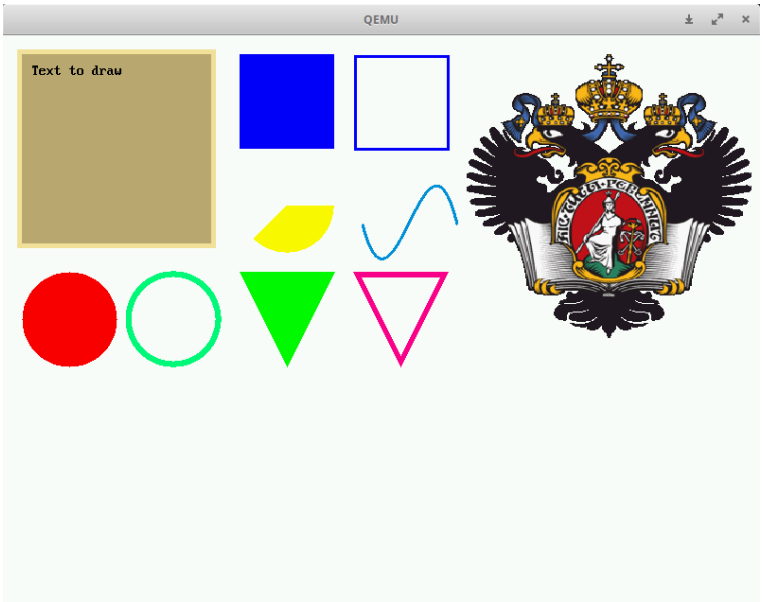

Рис. 10: Результат запуска примера, демонстрирующего примитивы, на виртуальной машине QEMU.

На рисунке 10 показано окно, которое вывелось посредством виртуальной машины на локальном компьютере. Видно, что поддерживаются такие элементы, как геометрические примитивы, текст и картинки.

На данный пример потребовалось следующее количество памяти:

- ROM: 277 Кб
  - .text 277 Кб

- RAM: 185 Кб
  - Heap: 172 Кб
  - .data 1 Кб
  - .bss 12 Кб

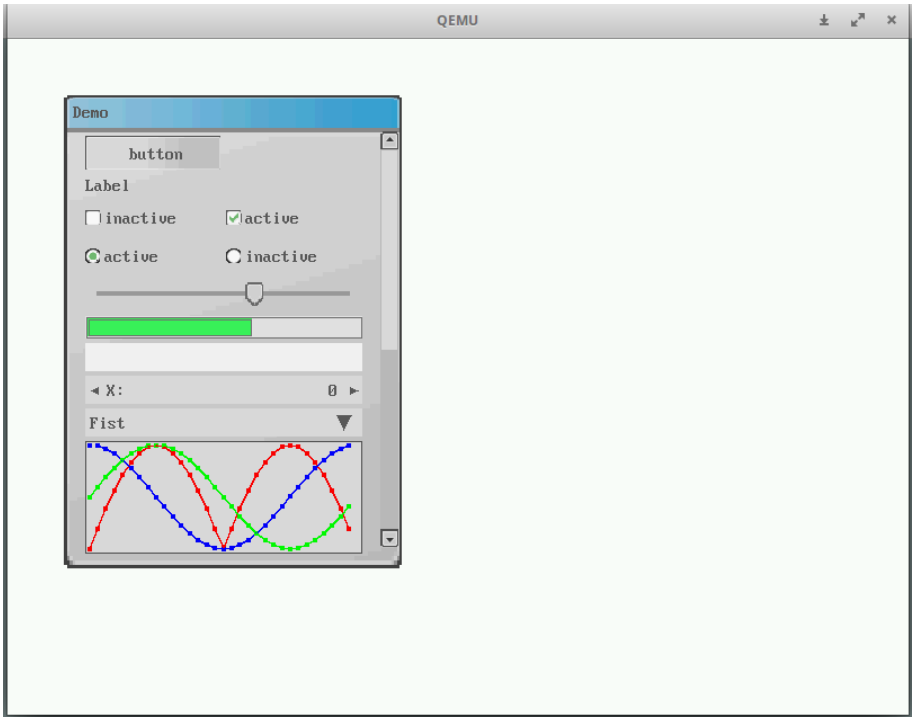

Рис. 11: Результат запуска примера, демонстрирующего виджет, на виртуальной машине QEMU.

На рисунке 11 показан виджет, который состоит из таких примитивов как прямоугольники и кривые линии, а также использует возможности графической библиотеки nuklear по созданию заголовков, полос прокрутки, слайдеров, кнопок и т.д., которые используют картинки в качестве фона. На данный пример потребовалось следующее количество памяти:

- ROM: 290 Кб
  - .text 290 Кб
- RAM: 583 Кб
  - Heap: 569 Кб
  - .data 1 Кб
  - .bss 13 Кб

5.2. Апробация на STM32F746G Discovery

Отладочная плата STM32F746G Discovery имеет следующие характеристики:

- 1 МБ Flash памяти и 340 КБ RAM памяти;
- 4,3” TFT-LCD дисплей, разрешение 480x272;
- на базе ядра ARM® Cortex®-M7;
- 128 Мбит Quad-SPI Flash память;
- 128 Мбит SDRAM (доступно 64 Мбит).

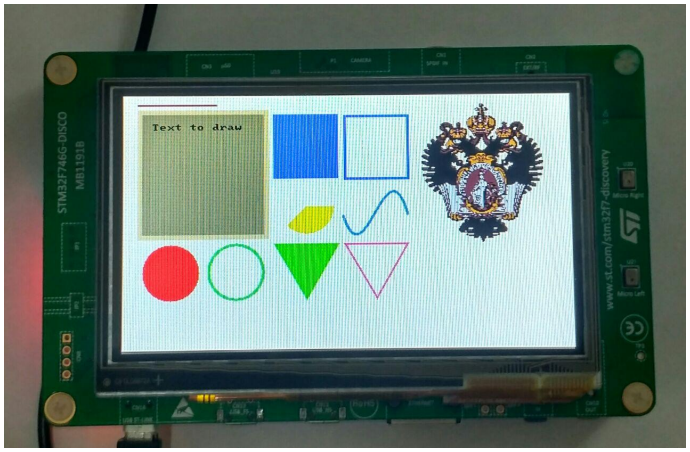

Рис. 12: Результат запуска примера, демонстрирующего примитивы, на плате STM32F746G.

На рисунке 12 показан результат запуска первого примера на плате STM32F746G. Можно увидеть, что результат подобен результату запуска на виртуальной машине QEMU (рисунок 10): сохранились все элементы, цвета и расположения. Для данного пример было затрачено следующее количество памяти:

- ROM: 252 Кб
  - .text 252 Кб
- RAM: 185 Кб
  - Heap: 173 Кб
  - .data 1 Кб
  - .bss 12 Кб

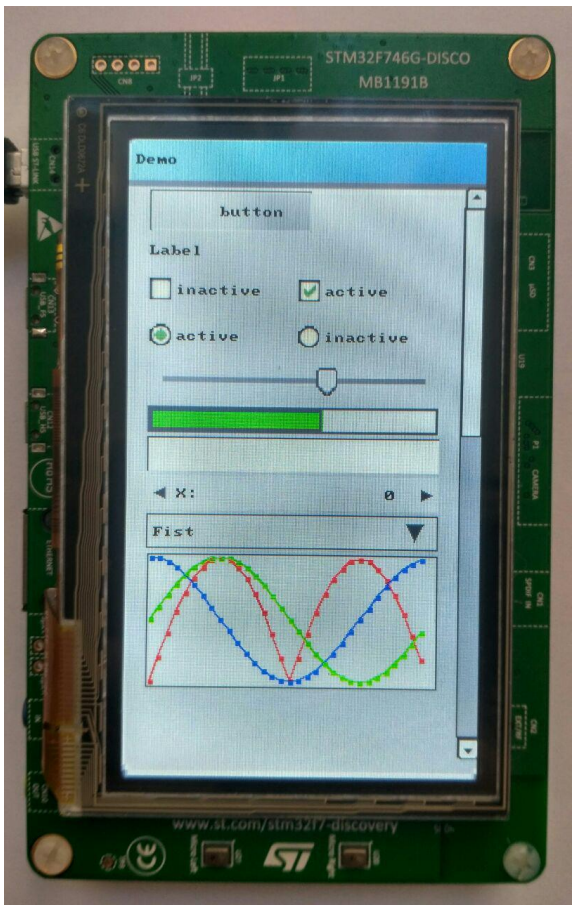

Рис. 13: Результат запуска примера, демонстрирующего виджет, на плате STM32F746G.

Для запуска второго примера, результаты которого изображены на рисунке 13, был применен поворот экрана, чтобы нарисовать виджет во весь экран в более подходящих пропорциях. Для второго примера потребовалось следующее количество памяти:

- ROM: 259 Кб
  - .text 259 Кб
- RAM: 583 Кб
  - Heap: 569 Кб
  - .data 1 Кб
  - .bss 13 Кб

По результатам апробации видно, что реализованная графическая подсистема может работать на различных устройствах с различными характеристиками.

Заключение

В данной работе были выполнены следующие задачи:

- Произведен обзор существующих архитектур (X Window System, Wayland, Nano-X, DirectFB), спецификации API (OpenGL, Vulkan, Glide) и графических библиотек (Nuklear, STML, NanoVG, ImGui, TIGR).
- Разработана архитектура конфигурируемой графической подсистемы.
- Выполнена реализация архитектуры на языке C, произведена интеграция со встраиваемой ОС Embox.
- Произведена апробация на виртуальной машине QEMU и на отладочной плате STM32F746G Discovery.

Список литературы

[1] Barrett Sean. Single-file public-domain/open source libraries with minimal dependencies. — URL: https://github.com/nothings/single_file_libs (online; accessed: 19.03.2017).

[2] Cornut Omar. Bloat-free Immediate Mode Graphical User interface for C++ with minimal dependencies. — URL: https://github.com/ocornut/imgui (online; accessed: 07.03.2017,).

[3] Embox. Modular and configurable OS kernel for embedded applications. — URL: https://github.com/embox/embox (online; accessed: 20.05.2017).

[4] Fast Light Toolkit. — URL: http://www.fltk.org/index.php (online; accessed: 06.03.2017,).

[5] Haerr Gregory. The Microwindows Project. Enabling graphical applications on embedded Linux systems. — URL: http://www.microwindows.org/MicrowindowsPaper.html (online; accessed: 24.02.2017).

[6] Mettke Micha. A single-header ANSI C gui library. — URL: https://github.com/vurtun/nuklear (online; accessed: 14.03.2017,).

[7] Microsoft. Retained Mode Versus Immediate Mode. — URL: https://msdn.microsoft.com/en-us/library/windows/desktop/ff684178(v=vs.85).aspx (online; accessed: 10.05.2017).

[8] Mitton Richard. TIRG Overview. — URL: https://bitbucket.org/rmitton/tigr/overview (online; accessed: 21.04.2017).

[9] Mononen Mikko. nanoVG. — URL: https://github.com/memononen/nanovg (online; accessed: 06.03.2017,).

[10] QEMU. QEMU—The fast processor emulator. — URL: http://www.qemu.org/ (online; accessed: 20.04.2017).

[11] STMicroelectronic. 32F746GDISCOVERY—Discovery kit with STM32F746NG MCU. — URL: http://www.st.com/en/evaluation-tools/32f746gdiscovery.html.

[12] Wayland. Wayland Architecture. — URL: https://wayland.freedesktop.org/architecture.html (online; accessed: 16.10.2016).

[13] Wiki Embedded Linux. DirectFB. — URL: http://elinux.org/DirectFB (online; accessed: 24.12.2016).

[14] Wikipedia. Framebuffer. — URL: https://en.wikipedia.org/wiki/Framebuffer (online; accessed: 22.10.2016).

[15] Wikipedia. List of platform-independent GUI libraries. — URL: https://en.wikipedia.org/wiki/List_of_platform-independent_GUI_libraries (online; accessed: 03.03.2017).

[16] Wikipedia. List of rendering APIs. — URL: https://en.wikipedia.org/wiki/List_of_rendering_APIs (online; accessed: 28.02.2017).

[17] Wikipedia. List of widget toolkits. — URL: https://en.wikipedia.org/wiki/List_of_widget_toolkits (online; accessed: 22.03.2017).

[18] Wikipedia. X Window System. — URL: https://en.wikipedia.org/wiki/X_Window_System (online; accessed: 12.10.2016).