Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

Отчёт по лабораторной работе

по курсу «Проектирование ОС и компонентов»

по теме «SRS спецификация для Fuchsia OS»

Выполнил студент гр. 13541/2:
Ерниязов Т.Е.

Проверил преподаватель:
Душутина Е. В.

Санкт-Петербург
2019 г.

Содержание

1 Введение **2**
1.1 Назначение . . . 2
1.2 Область применения . . . 2
1.3 Определения, акронимы и сокращения . . . 2
1.4 Ссылки . . . 3

2 Общее описание **3**
2.1 Перспектива изделия . . . 3
2.2 Архитектура и интерфейсы системы . . . 3
2.2.1 Пользовательский режим . . . 4
2.2.2 Режим ядра . . . 9
2.3 Функции изделия . . . 10
2.3.1 Интерфейсы пользователя . . . 10
2.3.2 Работа с оборудованием . . . 10
2.3.3 Доступ к данным . . . 10
2.3.4 Безопасность данных . . . 11
2.3.5 Многозадачность . . . 11
2.3.6 Совместная работа за компьютером . . . 11
2.3.7 Обработка аварийных ситуаций . . . 11
2.3.8 Сетевые ресурсы . . . 11
2.3.9 Автоматизация действий . . . 11
2.3.10 Поддержиываемые устройства . . . 11

1 Введение

1.1 Назначение

Данная спецификация требований программного обеспечения создана в рамках курса "Проектирование ОС и компонентов"для ознакомления с операционной системой Fuchsia OS, выработкой навыков ведения документации для программного обеспечения, а также ознакомлением со стандартом IEEE 830-1998

Документ иллюстрирует цель и полную спецификацию для разработки продукта. В нем также приведены системные ограничения, интерфейсы и взаимодействие с другими внешними приложениями.

1.2 Область применения

Fuchsia OS - это разрабатываемая, объектно-ориентированная, открытая операционная система реального времени с сильной ориентацией на облачные технологии. Эта операционная система рассчитана на все виды устройств, но основное ее назначение - замена Android (отказ от ядра Linux).

Концептуальные принципы:

- **Совместимость.** Так как Fuchsia OS является заменой операционной системы Android (а также Chrome OS), то данная система может работать с Android приложениями, чтобы обеспечить более безболезненный переход с одной операционной системы на другую. Более того, Fuchsia OS позволяет запускать любые приложения написанные с помощью Flutter. Таким образом до выхода операционной системы уже создана поддержка в виде множества нативных приложений.
- **Универсальность.** Данная операционная система разрабатывается для всех видов устройств. На данный момент Fuchsia предлагает два вида: новый мобильно-ориентированный дизайн под именем Armadillo и более привычный для десктопа Capybara.
- **Кроссплатформенность.** С помощью облачных технологий Fuchsia OS сохраняет весь контекст, который в последствии может быть открыт на другом устройстве для продолжения работы.
- **Операционная система реального времени.** Fuchsia является операционной системой реального времени, что позволяет ее применять для мобильных устройств, умных устройств и т.д.

Fuchsia — операционная система для всех типов устройств одновременно, поэтому ее целевой аудиторией является любой пользователь. Она может работать на смартфонах, планшетах, компьютерах, системах умного дома и даже на микроконтроллерах (если снять все лишнее и оставить только ядро LK). Ее интерфейс может растягиваться, сжиматься и произвольно изменять геометрию. Так как данная операционная система считается приемников Android, то все приложения для Android будут также доступны и для Fuchsia.

Большим преимуществом Fuchsia OS является легкое переключение между устройствами за счет использования облачного хранилища куда дублируются все: данные приложений, сами приложения (а точнее, компоненты), документы, настройки и т.д.

1.3 Определения, акронимы и сокращения

Определения и термины:

- Ledger — это распределенная система хранения для Fuchsia. Приложения используют Ledger либо напрямую, либо через примитивы синхронизации состояний, предоставляемые модульной структурой, которые основаны на внутреннем Ledger.
- Realm — контейнер для набора компонентов, который предоставляет способ управления их жизненным циклом и предоставления им услуг. Все компоненты в среде получают доступ к (подмножеству) службам среды.
- Zircon - это микроядро и компоненты пользовательского пространства самого низкого уровня (среда выполнения драйверов, драйверы ядра, libc и т. Д.) в ядре Fuchsia.
- Netstack - Реализация TCP, UDP, IP и связанных сетевых протоколов для Fuchsia.

- Banjo - это язык для определения протоколов, которые используются для связи между драйверами. Он отличается от FIDL тем, что он определяет ABI для драйверов, чтобы использовать их для вызова друг друга, а не протокол IPC.
-

Акронимы:

- LK – Little Kernel (LK) - это встроенное ядро, которое сформировало ядро Zircon Kernel. LK больше ориентирован на микроконтроллеры и не поддерживает MMU, пространство пользователя, системные вызовы - функции, добавленные Zircon.
- VMAR – Virtual Memory Address Range - это объект ядра Zircon, который контролирует, где и как VMO может быть отображен в адресное пространство процесса.
- VMO – Virtual Memory Object - это объект ядра Zircon, представляющий коллекцию страниц, которые можно читать, записывать, отображать в адресное пространство процесса или совместно использовать с другим процессом, передавая дескриптор через канал.
- FVM – Fuchsia Volume Manager - менеджер разделов, предоставляющий динамически распределенные группы блоков, известные как срезы, в адресное пространство виртуальных блоков. Разделы FVM предоставляют блочный интерфейс, позволяющий файловым системам взаимодействовать с ним способом, в значительной степени совместимым с обычным блочным устройством.
- FIDL – The Fuchsia Interface Definition Language - это язык для определения протоколов, которые обычно используются в каналах. FIDL не зависит от языка программирования и имеет привязки для многих популярных языков, включая C, C ++, Dart, Go и Rust. Этот подход позволяет системным компонентам, написанным на разных языках, взаимодействовать беспрепятственно.
- ОСРВ - Операционная система реального времени.

1.4 Ссылки

[1] https://9to5google.com/ [электронный ресурс] - информационный портал, где выкладываются новые новости о разрабатываемой ОС (дата обращения 09.04.2019)

[2] https://fuchsia.googlesource.com/ [электронный ресурс] - официальный репозиторий с открытым исходным кодом операционной системы (дата обращения 09.04.2019)

[3] https://github.com/leisim/awesome-fuchsia [электронный ресурс] - репозиторий, на котором проведен анализ исходного кода (дата обращения 09.04.2019)

2 Общее описание

2.1 Перспектива изделия

Fuchsia OS в первую очередь должна заменить операционную систему Android для мобильных устройств. В настоящее время, так как Android не является операционной системой реального времени, то его ставят в пару ОСРВ на мобильное устройство. Fuchsia является операционной системой реального времени, поэтому является полностью автономной.

Помимо настольных компьютеров и мобильных устройств система может так же ставится и на более специфичных устройствах, таких как микроконтроллеры. Fuchsia состоит из модулей, а поэтому ненужные модули можно попросту отключить, оставив только микроядро Little Kernel. На данный момент, Fuchsia Os поддерживает ARM и x86-64 процессоры.

2.2 Архитектура и интерфейсы системы

Архитектура Fuchsia OS включает в себя два основных уровня — пользовательский режим (user mode) и режим ядра (kernel mode). Так как Fuchsia это операциооная система имеющая микроядерную архитектуру, то в микроядре находится только минимальная функциональность. В Google операционную систему

разбивают на 4 уровня: Zircon, Garnet, Peridot и Topaz. Только некоторая часть уровня Zircon находится в ядре, все остальные компоненты расположены в пространстве пользователя. Каждый из уровней наслаивается на предыдущий:

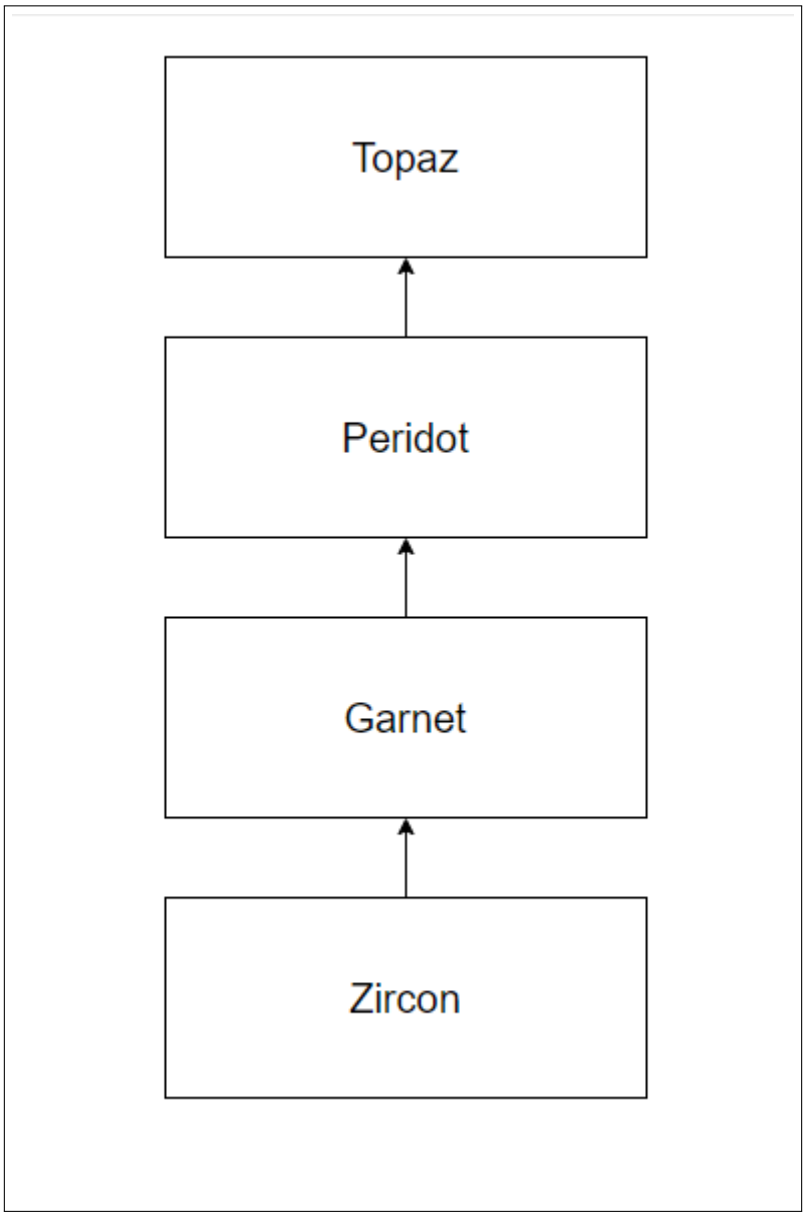

Рис. 1: Слои Fuchsia OS

2.2.1 Пользовательский режим

Пользовательский режим представляет собой уровень поддержки приложений. Именно в этой части операционной системы сторонние производители программного обеспечения могут выполнять системные вызовы готовых интерфейсов API и объектно-ориентированных компонентов. Все приложения и службы устанавливаются на уровне пользовательского режима. В этом режиме работают уровни Topaz, Garnet, Peridot и часть уровня Zircon.

Topaz Пользовательские приложения в основном взаимодействуют именно с этим уровнем. Topaz расширяет функциональность системы, реализуя интерфейсы, определенные нижележащими уровнями. Для разработки приложений с пользовательским интерфейсом рекомендуется использовать фреймворк Flutter. Flutter - это функционально-реактивный фреймворк для разработки пользовательского интерфейса, оптимизированный для Fuchsia, который используется многими компонентами системы. Flutter также работает на множестве других платформ, включая Android и iOS. Сама Fuchsia не требует использования какого-либо конкретного языка программирования или фреймворков для пользовательского интерфейса.

FIDL FIDL - это язык, используемый для описания протоколов межпроцессного взаимодействия (IPC), используемых программами, работающими в операционной системе Fuchsia. FIDL поддерживается набором инструментов (компилятором) и библиотеками поддержки времени выполнения (связывания), чтобы помочь разработчикам эффективно использовать IPC.

Fuchsia в значительной степени опирается на IPC, поскольку она имеет архитектуру микроядра, в которой большинство функций реализовано в пространстве пользователя вне ядра, включая привилегированные компоненты, такие как драйверы устройств.

Peridot Peridot - это структура для сложного, интеллектуального и распределенного взаимодействия пользователей.

Приложения, явно не предназначенные для взаимодействия (и, возможно, реализованные на разных языках программирования), загружаются эфемерно и динамически составляются для работы в общем контексте. Peridot управляет жизненным циклом приложения, ресурсами и иерархией представления.

Состояние Peridot прозрачно синхронизируется между пользовательскими устройствами с помощью распределенной системы хранения в автономном режиме через Ledger.

Каждое приложение, запущенное от имени определенного пользователя, имеет отдельное хранилище данных, предоставляемое и управляемое Ledger, и переданное клиентскому приложению средой Fuchsia через контекст компонента.

Хранилище данных для конкретной комбинации компонент / пользователь является частным - недоступно для других приложений того же пользователя и недоступно для других пользователей того же приложения.

Каждое хранилище данных прозрачно синхронизируется между устройствами его пользователя через облачного провайдера. Любые операции с данными выполняются в автономном режиме, без координации с облаком. Если одновременные изменения приводят к конфликту данных, конфликт разрешается с помощью настраиваемой приложением политики слияния. По умолчанию в качестве облачного хранилища используется Firestore, однако, есть возможность заменить его на любой другой.

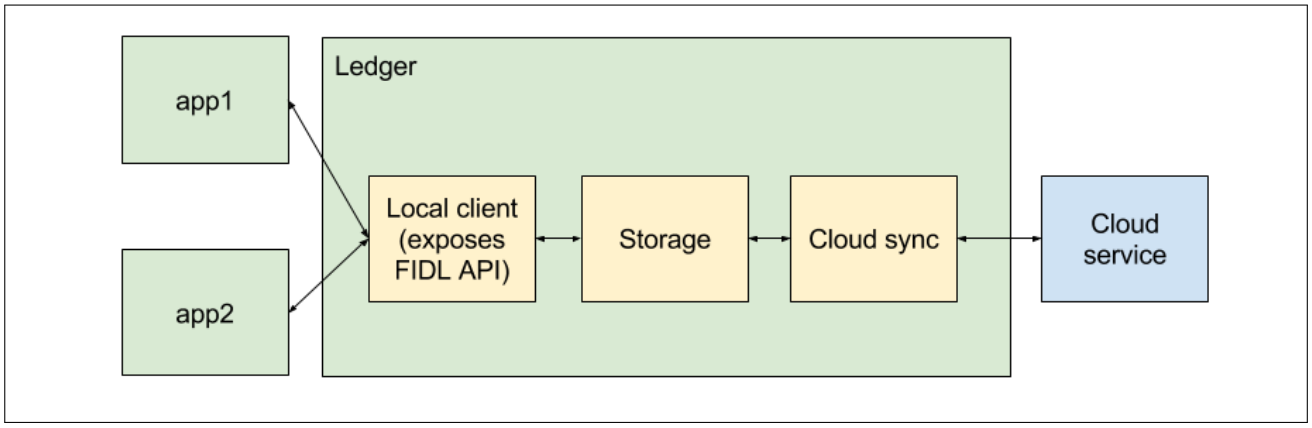

Рис. 2: Структура Ledger

Modular - это фреймворк, который управляет пользовательским интерфейсом, объединяя пользовательский интерфейс, данные и пользователей из разнообразного набора компонентов в логические и визуальные контейнеры, называемые Stories.

Modular определяет классы компонентов для расширения взаимодействия с пользователем и предоставляет программные примитивы для композиции компонентов, связи, делегирования задач, управления состоянием и совместного использования данных.

Modular поддерживает программное обеспечение, написанное на любом языке (например, Flutter, C ++) для любой среды выполнения, поддерживаемой Fuchsia, при условии, что это компонент Fuchsia.

Фреймворк Modular связывается с компонентами, которые она запускает, через FIDL, стандартный механизм IPC для Fuchsia.

Фреймворк определяет несколько различных классов компонентов, которые могут быть реализованы разработчиками для расширения поведения пользователей:

- Modules - это компоненты, которые отображают пользовательский интерфейс и визуально состоят из Story (логический контейнер для набора модулей).
- Agents - это компоненты, которые работают в фоновом режиме для предоставления услуг и данных модулям и другим агентам.
- Shells управляют пользовательским интерфейсом системы и обеспечивают взаимодействие с пользователем.
- EntityProviders - это компоненты, которые обеспечивают доступ к объекту данных (сущностям), которые совместно используются компонентами, работающими в Modular.

Garnet Garnet - это первый слой Fuchsia над ядром. Он содержит множество низкоуровневых вещей, которые нужны каждой ОС, включая драйверы устройств (сеть, графика и т. Д.) и установку программного обеспечения.

Слой Garnet не предоставляет полноценного пользовательского интерфейса конечного пользователя. Вместо этого он предоставляет услуги, которые обеспечивают основу для создания безопасных, производительных, многопроцессорных пользовательских интерфейсов. В совокупности эти службы известны как «Mozart».

Эти услуги включают в себя:

- Scenic, графический движок Fuchsia. Scenic предоставляет граф сцены с сохранением режима, который позволяет составлять и визуализировать графические объекты из нескольких процессов в единой среде освещения.
- View Manager поддерживает иерархическое встраивание клиентских модулей и отвечает за распространение информации о макете, диспетчеризацию входных событий и поддержку модели фокуса пользователя.
- Input отвечает за обнаружение доступных устройств ввода и позволяет клиентам регистрироваться на события с этих устройств.

Scenic - это сервис Garnet, который объединяет графические объекты из нескольких процессов в общий граф сцены. Эти объекты визуализируются в единой среде освещения (для дисплея или другой цели рендеринга); это означает, что объекты могут отбрасывать тени или отражать свет друг на друга, даже если исходные процессы не знают друг друга.

Обязанности Scenic:

- Композиция: Scenic предоставляет трехмерный граф сцены с сохранением режима, который содержит контент, который независимо генерируется и связывается вместе своими клиентами. Композиция позволяет беспрепятственно смешивать графическое содержимое отдельно реализованных компонентов пользовательского интерфейса.
- Анимация: Scenic переоценивает любые изменяющиеся во времени выражения в моделях перед рендерингом каждого кадра, тем самым обеспечивая анимацию свойств модели без дальнейшего вмешательства со стороны клиентов. Выгрузка анимации в Scenic обеспечивает плавные переходы без рывков.
- Рендеринг: Scenic визуализирует свой граф сцены, используя Escher, библиотеку рендеринга, построенную на Vulkan, которая применяет реалистичное освещение и тени ко всей сцене.
- Планирование: Scenic планирует обновления сцены и анимации, чтобы предвидеть и соответствовать задержке представления цели рендеринга и интервалу обновления.
- Диагностика: Scenic предоставляет диагностический интерфейс, который помогает разработчикам отлаживать свои модели и измерять производительность.

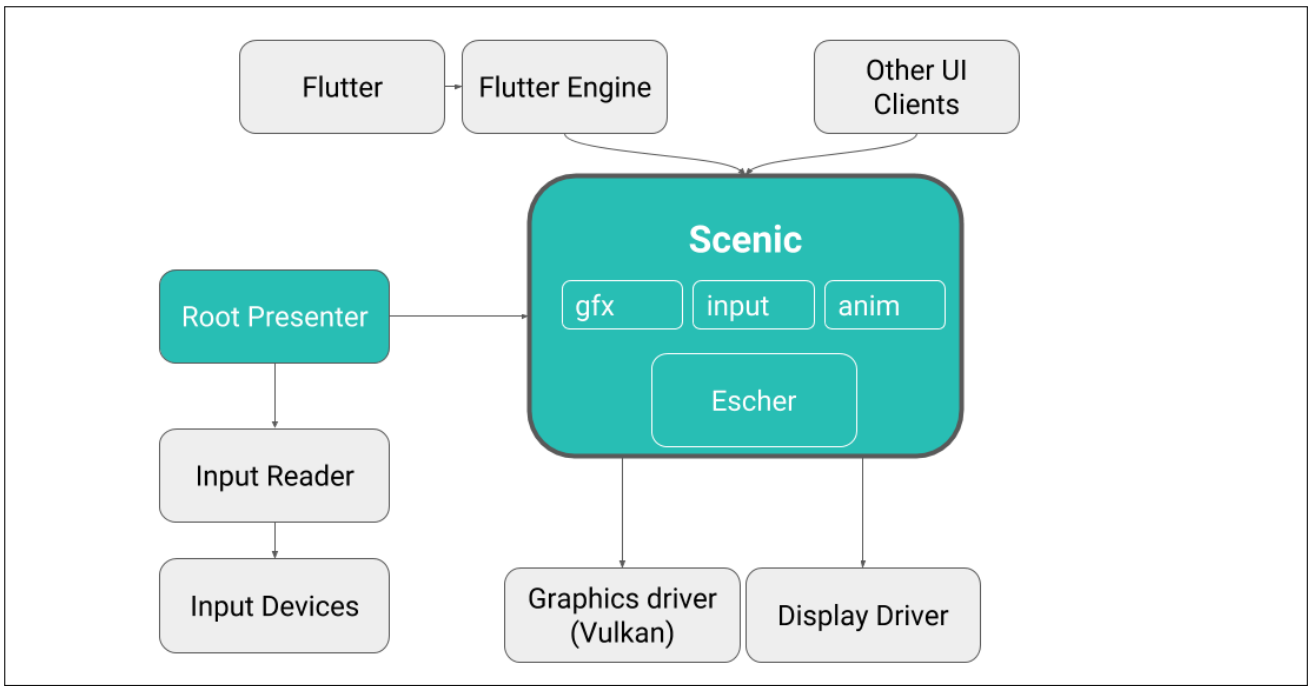

Рис. 3: Взаимодействие с Scenic

API Scenic позволяет любому клиенту вставить свой интерфейс в глобальный граф сцены. Нижний уровень Flutter, называемый Flutter Engine, содержит код, отвечающий за связь со Scenic. Scenic является клиентом графического драйвера Vulkan и системного Display Driver.

Amber - это система обновлений с целью обновления всех компонентов, работающих в системе Fuchsia, независимо от того, является ли этот компонент ядром, загрузчиком, системной службой, приложением и т. Д.

Zirkon Zircon является основной платформой, на которой работает Fuchsia OS. Zircon состоит из микроядра, а также небольшого набора сервисов, драйверов и библиотек пользовательского пространства, необходимых для загрузки системы, общения с оборудованием, загрузки процессы пользователя и их запуск и т.д.

Platform Bus - это фреймворк для управления драйверами в Zircon. Platform bus содержит несколько типов драйверов:

- Драйвер шины, который управляет шиной. Это универсальный драйвер без аппаратной функциональности.
- Board driver, который является первым драйвером, загружаемым драйвером шины. Драйвер платы содержит всю информацию о платформе, необходимую для шины, и контролирует, какие другие драйверы будут загружены Platform Bus.
- Platform device drivers являются основой для драйверов более высокого уровня в Zircon. Эти драйверы обеспечивают самый низкий уровень поддержки определенной функции, такой как USB, eMMC или NAND-хранилище, и т. Д., Причем драйверы более высокого уровня загружаются поверх этого.
- Protocol implementation drivers, такие как GPIO или I2C, которые обеспечивают поддержку протокола в качестве ресурсов для драйвера шины.
- Proxy client driver - это драйвер, который реализует поддержку клиента для поставщика или протокола SOC.
- Platform proxy driver является дополнением к драйверу шины.

В Zirkon находятся драйвера для работы с блочными устройствами. Драйверы устройств Fuchsia, как и другие драйверы в системе, реализованы в виде сервисов пользовательского пространства, доступных через IPC. Программы, использующие блочные устройства, будут иметь один или несколько дескрипторов этих базовых драйверов. Подобно клиентам файловой системы, которые могут отправлять запросы «чтения» или «записи» на серверы путем кодирования этих запросов в сообщениях RPC, программы

могут выступать в качестве клиентов для блокировки устройств и могут передавать сообщения RPC на «хост устройства». Затем процесс devhost преобразует эти запросы в понимаемые драйвером «транзакции ввода / вывода», где они фактически передаются конкретному драйверу блочного устройства и, в конечном итоге, реальному оборудованию.

Драйверы блочных устройств часто отвечают за захват больших порций памяти и постановку в очередь запросов к конкретному устройству для «чтения» или «записи из» части памяти. К сожалению, вследствие передачи сообщений ограниченного размера из протокола RPC в «транзакцию ввода-вывода» часто требуется повторное копирование больших буферов для доступа к блочным устройствам.
Чтобы избежать этого узкого места в производительности, драйверы блочных устройств реализуют другой механизм для передачи операций чтения и записи: быстрый протокол на основе FIFO, который действует на совместно используемую VMO. Файловые системы (или любой другой клиент, желающий взаимодействовать с блочным устройством) могут получать FIFO от блочного устройства, регистрировать «буфер транзакций» и передавать дескрипторы VMO на блочное устройство. Вместо передачи сообщений «чтение» или «запись» с большими буферами клиент этого протокола может вместо этого отправить быстрое и легкое управляющее сообщение в FIFO, указывающее, что драйвер блочного устройства должен действовать непосредственно на уже зарегистрированную VMO.

В отличие от более распространенных монолитных ядер, файловые системы Fuchsia живут исключительно в пространстве пользователя. Они не связаны и не загружены с ядром; это просто процессы пользовательского пространства, которые реализуют серверы, которые могут отображаться как файловые системы. Как следствие, сами файловые системы Fuchsia могут быть легко изменены - изменения не требуют перекомпиляции ядра. Фактически, обновление до новой файловой системы Fuchsia может быть выполнено без перезагрузки.

Как и другие собственные серверы на Fuchsia, основной режим взаимодействия с сервером файловой системы достигается с помощью примитива дескриптора, а не системных вызовов. Ядро не знает о файлах, каталогах или файловых системах. Как следствие, клиенты файловой системы не могут напрямую запрашивать у ядра «доступ к файловой системе».

Эта архитектура подразумевает, что взаимодействие с файловыми системами ограничено следующим интерфейсом:
Сообщения отправляются по каналам связи, установленным на сервере файловой системы. Эти каналы связи могут быть локальными для клиентской файловой системы или удаленными. Процедура инициализации (которая, как ожидается, будет сильно настроена для каждой файловой системы; для сетевой файловой системы потребуется доступ к сети, для постоянных файловых систем может потребоваться доступ к блочному устройству, для файловых систем в памяти потребуется только механизм выделения новых временных страниц). В качестве преимущества этого интерфейса любые ресурсы, доступные через канал, могут выглядеть как файловые системы благодаря реализации ожидаемых протоколов для файлов или каталогов.

Текущие доступные файловые системы:

- MemFS: файловая система в памяти. MemFS используется для реализации запросов к временным файловым системам, таким как / tmp, где файлы полностью существуют в ОЗУ и не передаются на базовое блочное устройство. Эта файловая система в настоящее время также используется для протокола «bootfs», где большая виртуальная VMO только для чтения, представляющая коллекцию файлов и каталогов, разворачивается в доступные для пользователя Vnodes при загрузке.
- MinFS: постоянная файловая система. MinFS - это простая традиционная файловая система, которая способна постоянно хранить файлы. Как и MemFS, он широко использует слои VFS, упомянутые ранее, но в отличие от MemFS, он требует дополнительного дескриптора для блочного устройства (которое передается при запуске новому процессу MinFS).
- Blobfs: неизменяемая файловая система хранения пакетов с проверкой целостности. Blobfs - это простая плоская файловая система, оптимизированная для подписанных данных с однократной записью и только для чтения, таких как пакеты приложений. Кроме двух небольших предварительных условий (имена файлов, которые являются детерминированными, адресно-ориентированные хеши содержимого корня дерева Merkle дерева, для проверки целостности) и передачи знаний о размере файла Blobfs выглядит как типичная файловая система.
- ThinFS: файловая система FAT, написанная на Go. ThinFS - это реализация файловой системы FAT в Go. Это служит двойной цели: во-первых, доказательство того, что наша система на самом деле

модульная и способна использовать новые файловые системы, независимо от языка или времени выполнения. Во-вторых, он предоставляет механизм для чтения универсальной файловой системы, найденной в разделах EFI и многих USB-накопителях.

- Fuchsia Volume Manager - это «менеджер логических томов», который добавляет гибкость поверх существующих блочных устройств. Текущие функции включают возможность добавлять, удалять, расширять и сокращать виртуальные разделы. Чтобы сделать эти возможности возможными, внутренне fvm поддерживает физическое отображение на виртуальное от (виртуальные разделы, блоки) до (слайс, физический блок). Чтобы свести к минимуму накладные расходы на обслуживание, это позволяет разделам сокращаться / расти в виде кусков, называемых слайсами. Срез кратен собственному размеру блока. Помимо метаданных, остальная часть устройства делится на фрагменты. Каждый фрагмент либо свободен, либо принадлежит одному и только одному разделу. Если срез принадлежит разделу, то FVM сохраняет метаданные о том, какой раздел использует срез, и виртуальный адрес среза в этом разделе.

2.2.2 Режим ядра

В режиме ядра работает только микроядро уровня Zirkon. Микроядро предоставляет минимальный набор примитивов и механизмов для:

- управления памятью компьютерной системы, как физической так и виртуальной (выделение памяти процессам, обеспечение её изоляции/защиты);
- управления процессорным временем (сервисы для работы с потоками (нитями) процессов);
- управления доступом к устройствам ввода-вывода (открытие/закрытие доступа к портам ввода-вывода и MMIO-памяти устройств);
- осуществления межпроцессной коммуникации и синхронизации (англ. inter process communications, IPC).

Ядро Zircon предоставляет системные вызовы для управления процессами, потоками, виртуальной памятью, межпроцессным взаимодействием, ожиданием изменений состояния объекта и блокировками (через фьютексы).

Объекты ядра

- Handles (Дескрипторы) - это конструкции ядра, которые позволяют программам пользовательского режима ссылаться на объект ядра.
- System calls. Код пользовательского пространства взаимодействует с объектами ядра через системные вызовы и почти исключительно через дескрипторы.
- Kernel Object Ids. Идентификаторы объектов ядра. Каждый объект в ядре имеет «идентификатор объекта ядра» или «koid» для краткости.
- Jobs, Processes, and Threads. Потоки (Threads) представляют собой потоки выполнения в адресном пространстве, которое принадлежит процессу (Process), в котором они существуют. Процессы принадлежат джобам (Jobs), которые определяют различные ограничения ресурсов. Джобы принадлежат родительским джобам, вплоть до корневого.
- Передача сообщений: Sockets and Channels. И сокеты, и каналы являются объектами IPC, которые являются двунаправленными и двусторонними. Сокеты ориентированы на поток, и данные могут записываться или считываться из них в единицах одного или нескольких байтов. Каналы ориентированы на дейтаграммы и имеют максимальный размер сообщения может задаваться.
- Сигналы объектов. Объекты могут иметь до 32 сигналов, которые представляют часть информации об их текущем состоянии.
- События, Пары событий. Событие - это простейший объект, не имеющий другого состояния, кроме своей коллекции активных сигналов. Пара событий - это одна из пары событий, которые могут сигнализировать друг другу.
- Области адресов виртуальной памяти (VMAR) предоставляют абстракцию для управления адресным пространством процесса. Во время создания процесса дескриптор корневого VMAR предоставляется создателю процесса.
- Futexes - это примитивы ядра, используемые с атомарными операциями в пространстве пользователя для реализации эффективных примитивов синхронизации.

Управление потоками По сути, на каждом логическом процессоре в машине работает планировщик. Эти планировщики работают независимо и используют IPI (межпроцессорные прерывания) для координации. Однако каждый процессор отвечает за планирование потоков, которые на нем работают.

Каждый ЦП имеет свой собственный набор очередей приоритетов. Один для каждого уровня приоритета в системе, в настоящее время 32. Это очереди FIFO, а не структура данных, известная как очередь с приоритетами. В каждой очереди есть упорядоченный список запускаемых потоков, ожидающих выполнения. Когда приходит время запуска нового потока, планировщик просто просматривает очередь с наибольшим номером, которая содержит поток, вытаскивает заголовок из этой очереди и запускает этот поток. Если в очередях нет потоков для запуска, вместо этого будет запущен свободный поток

Каждому потоку назначается одинаковый размер временного интервала, когда он выбран для запуска. Если он использует весь свой временной интервал, он будет повторно вставлен в конец соответствующей очереди приоритетов. Однако, если у него есть некоторый временной интервал, оставшийся от предыдущего запуска, он будет вставлен в начало очереди с приоритетами, чтобы он мог возобновиться как можно быстрее. Когда он снова поднят, он будет работать только для оставшейся части предыдущего временного интервала.

Когда планировщик выбирает новый поток из очереди приоритетов, он устанавливает таймер вытеснения ЦП либо для полного временного интервала, либо для остатка предыдущего временного интервала. Когда этот таймер сработает, планировщик остановит выполнение в этом потоке, добавит его в соответствующую очередь, выберет другой поток и начнет заново.

Если поток блокирует ожидание общего ресурса, он удаляется из своей приоритетной очереди и помещается в очередь ожидания для общего ресурса. Когда он разблокирован, он будет повторно вставлен в соответствующую приоритетную очередь соответствующего ЦП, и, если для его запуска оставался временный интервал, он будет добавлен в начало очереди для ускоренной обработки.

2.3 Функции изделия

2.3.1 Интерфейсы пользователя

Механизм взаимодействия человека с программой называют интерфейсом пользователя. В это понятие включают внешний вид программы на экране, основные принципы управления и даже конкретные команды. Иными словами, интерфейс пользователя — это способ представления программ. В операционной системе Fuchsia OS интерфейс пользователя различается для программ, в зависимости от устройства, в контексте которой пользователь находится в данный момент. На данный момент Fuchsia предлагает два вида: новый мобильно-ориентированный дизайн под именем Armadillo и более привычный для десктопа Capybara.

2.3.2 Работа с оборудованием

Это самая важная функция операционной системы. Для каждого устройства компьютера нужен драйвер, небольшая программа, встраиваемая в операционную систему. Именно драйвер и представляет устройство. Драйвер получает от программ стандартизированные команды и переводит их в специфические инструкции устройства. Операционная система контролирует наличие драйверов устройств и корректность их работы.

2.3.3 Доступ к данным

Постоянные данные на компьютере обычно хранятся на жестком диске. Операционная система обеспечивает доступ к данным, а также контролирует правильность и надежность операций чтения и записи. Аналогично организуется доступ к другим носителям (гибким дискам, компакт-дискам и прочим). Элементы данных при этом представляются в удобной для человека форме. Операционная система имеет возможность сохранять контекст, в котором находился пользователь. Таким образом можно ввести работу на обычном компьютере, а потом продолжить сессию с мобильного устройства ничего не потеряв. Весь контекст сохраняется в облаке, для этого только нужно подключение к серверу. По умолчанию для этих целей используется Firestore, однако, существует возможность его сменить. сохранять контекст сессии и загружать его после переключения между операционными системами.

2.3.4 Безопасность данных

В отличие от Android и Chrome OS, Fuchsia построена на собственном микроядре Zircon, а не на Linux. Zicron основано на модели мандатных ссылок (capability-based); в отличие от модели прав, где ресурсы существуют в глобальном пространстве и возможность доступа к ресурсам определяется наличием у процесса определённых прав, в Zicron доступ к ресурсам осуществляется через специальные ссылки (handles), что делает систему более устойчивой и безопасной.

2.3.5 Многозадачность

В Fuchsia OS несколько программ могут работать на компьютере одновременно. Операционная система воспринимает каждую работающую программу как процесс. Она контролирует переключение между задачами и выделение им ресурсов. Каждый процесс может самостоятельно обращаться к устройствам компьютера. Операционная система принимает запросы, выделяет программам запрошенные ресурсы и устраняет возникшие конфликты.

2.3.6 Совместная работа за компьютером

Персональным компьютером часто пользуются несколько человек. Операционная система контролирует права на доступ к данным и позволяет настроить рабочую среду по своему вкусу. Каждый пользователь может работать в соответствии с собственным представлением об удобстве.

2.3.7 Обработка аварийных ситуаций

И аппаратные, и программные средства не застрахованы от сбоев. Задача операционной системы — свести последствия таких сбоев к минимуму. В операционной системе Fuchsia ошибка отдельной программы не должна влиять на работу других активных программ и на саму операционную систему.

2.3.8 Сетевые ресурсы

Когда компьютер входит в состав сети, сетевые операции выполняются под контролем операционной системы. В первую очередь операционная система контролирует доступ к сети и обеспечивает безопасность работы. Она также определяет, какие внешние ресурсы доступны и какие ресурсы компьютера надо предоставить внешним потребителям.

2.3.9 Автоматизация действий

Повторяющиеся, однотипные, рутинные операции можно автоматизировать, например автоматически запускать программы и обслуживать компьютер. Обслуживание — это вспомогательные операции для проверки системы и поддержания ее работоспособности.

2.3.10 Поддержиываемые устройства

Разработчиками заявлено, что Fuchsia OS это универсальная операционная система. Однако, на данный момент ее полноценный запуск возможен только на следующих устройствах:

- Acer Switch Alpha 12
- Intel NUC (also this)
- Pixelbook