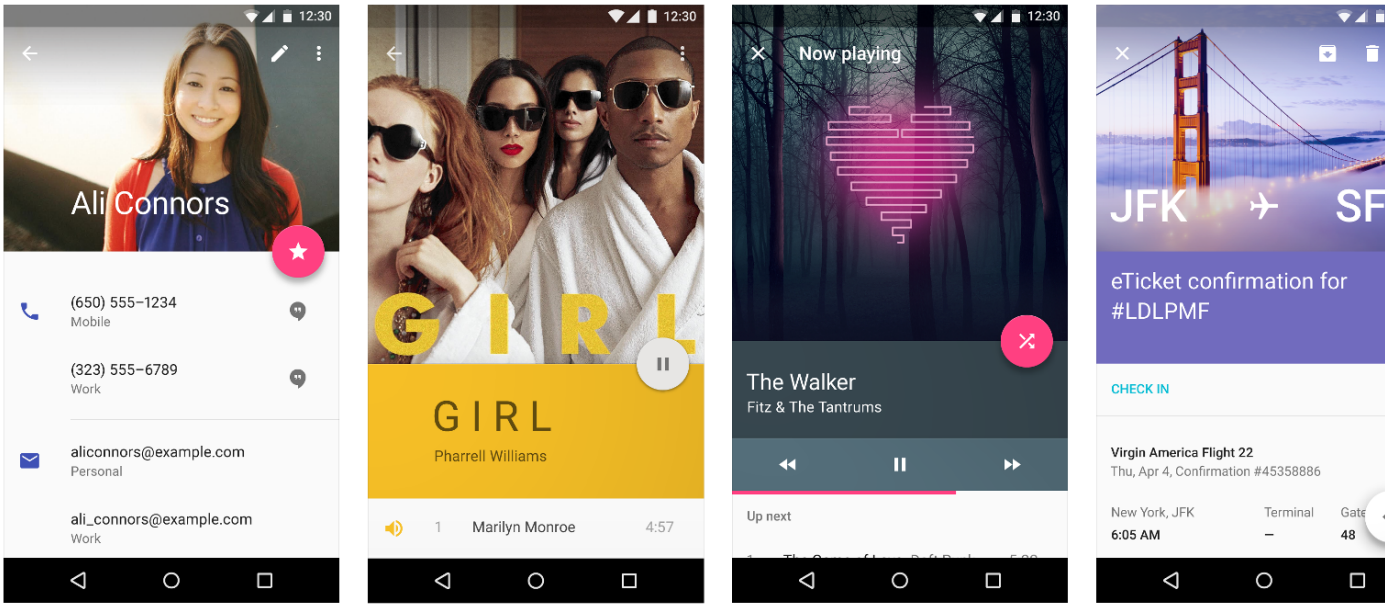
401,35
Рейтинг

Solar Security
Безопасность по имени Солнце

bugaevc 3 августа в 14:46 Разработка

Как работает Android, часть 1

Разработка под Android, Блог компании Solar Security



В этой серии статей я расскажу о внутреннем устройстве Android — о процессе загрузки, о содержимом файловой системы, о Binc Android Runtime, о том, из чего состоят, как устанавливаются, запускаются, работают и взаимодействуют между собой приложения Android Framework, и о том, как в Android обеспечивается безопасность.

Статьи серии:

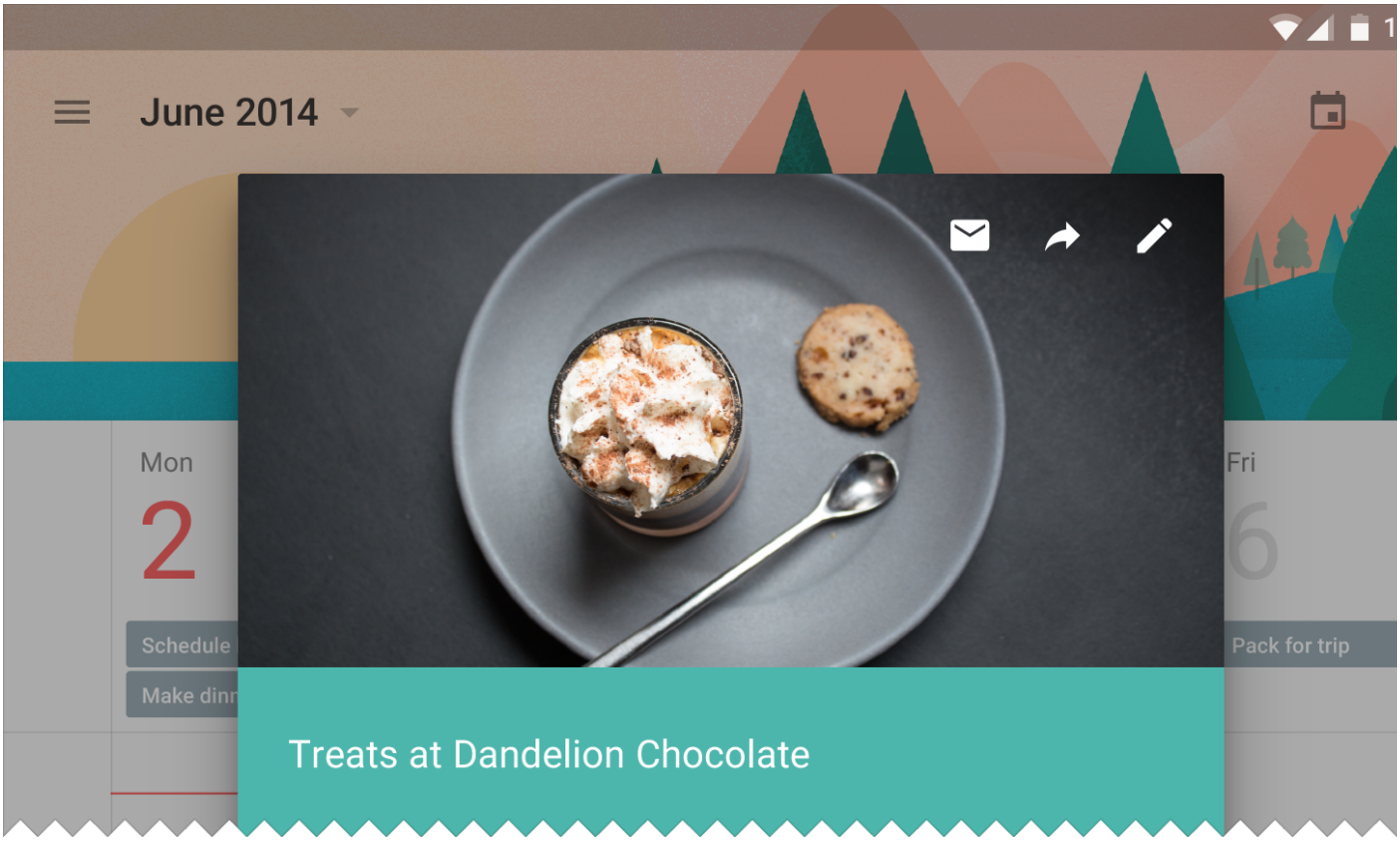
- Как работает Android, часть 1
- Как работает Android, часть 2
- Как работает Android, часть 3
- ...

Немного фактов

Android — самая популярная операционная система и платформа для приложений, насчитывающая больше двух *миллиардов* актив пользователей. На ней работают совершенно разные устройства, от «интернета вещей» и умных часов до телевизоров, ноутбуков автомобилей, но чаще всего Android используют на смартфонах и планшетах.

Android — свободный и открытый проект. Большинство исходного кода (который можно найти на https://source.android.com) распространяется под свободной лицензией Apache 2.0.

Компания Android Inc. была основана в 2003 году и в 2005 году куплена Google. Публичная бета Android вышла в 2007 году, а перва стабильная версия — в 2008, с тех пор мажорные релизы выходят примерно раз в год. Последняя на момент написания стабильна версия Android — 7.1.2 Nougat.



Android is Linux

По поводу такой формулировки было много споров, так что сразу поясню, что именно я имею в виду под этой фразой: Android осно на ядре Linux, но значительно отличается от большинства других Linux-систем.

Среди исходной команды разработчиков Android был Robert Love, один из самых известных разработчиков ядра Linux, да и сейчас компания Google остаётся одним из самых активных контрибьюторов в ядро, поэтому неудивительно, что Android построен на осно Linux.

Как и в других Linux-системах, ядро Linux обеспечивает такие низкоуровневые вещи, как управление памятью, защиту данных, поддержку мультипроцессности и многопоточности. Но — за несколькими исключениями — вы не найдёте в Android других привы компонентов GNU/Linux-систем: здесь нет ничего от проекта GNU, не используется X.Org, ни даже systemd. Все эти компоненты заменены аналогами, более приспособленными для использования в условиях ограниченной памяти, низкой скорости процессора минимального потребления энергии — таким образом, Android больше похож на встраиваемую (embedded) Linux-систему, чем на GNU/Linux.

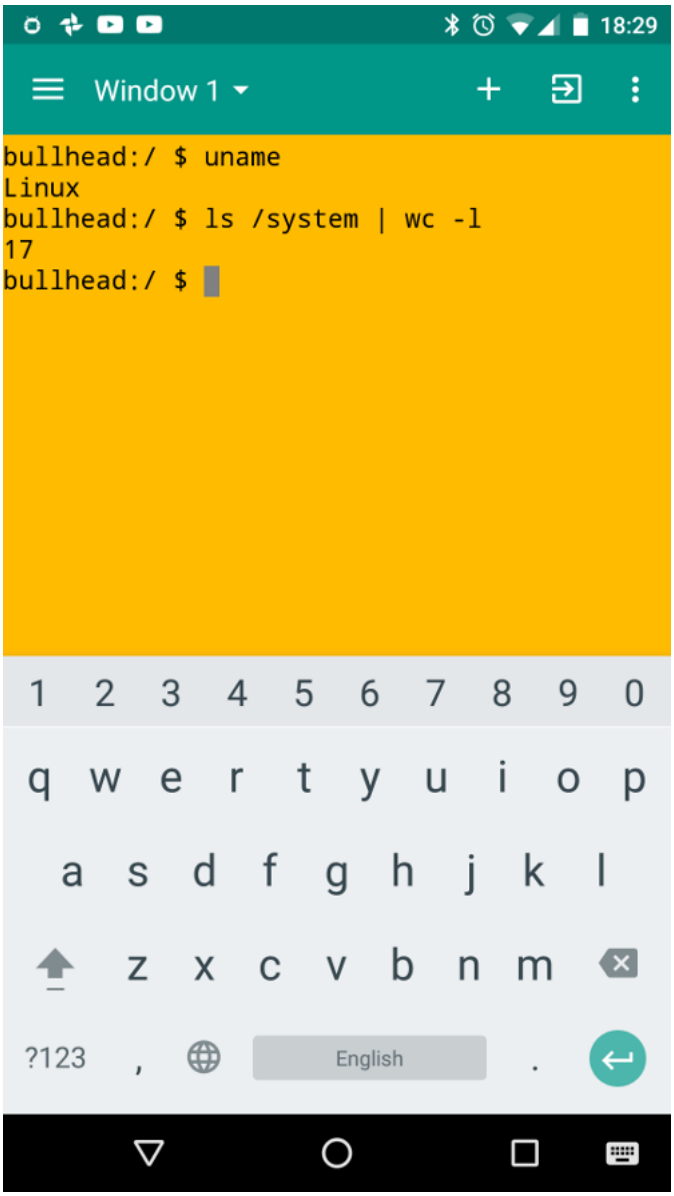
Другая причина того, что в Android не используется софт от GNU — известная политика «no GPL in userspace»:

> We are sometimes asked why Apache Software License 2.0 is the preferred license for Android. For userspace (that is, non-kernel) softw we do in fact prefer ASL 2.0 (and similar licenses like BSD, MIT, etc.) over other licenses such as LGPL.
>
> Android is about freedom and choice. The purpose of Android is promote openness in the mobile world, and we don't believe it's possibl predict or dictate all the uses to which people will want to put our software. So, while we encourage everyone to make devices that are o and modifiable, we don't believe it is our place to force them to do so. Using LGPL libraries would often force them to do just that.

Само ядро Linux в Android тоже *немного* модифицировано: было добавлено несколько небольших компонентов, в том числе ashmer (anonymous shared memory), Binder driver (часть большого и важного фреймворка Binder, о котором я расскажу ниже), wakelocks (управление спящим режимом) и low memory killer. Исходно они представляли собой патчи к ядру, но их код был довольно быстро добавлен назад в upstream-ядро. Тем не менее, вы не найдёте их в «обычном линуксе»: большинство других дистрибутивов отклю эти компоненты при сборке.

В качестве libc (стандартной библиотеки языка C) в Android используется не GNU C library (glibc), а собственная минималистичная реализация под названием bionic, оптимизированная для встраиваемых (embedded) систем — она значительно быстрее, меньше и менее требовательна к памяти, чем glibc, которая обросла множеством слоёв совместимости.

В Android есть оболочка командной строки (shell) и множество стандартных для Unix-подобных систем команд/программ. Во встраиваемых системах для этого обычно используется пакет Busybox, реализующий функциональность многих команд в одном исполняемом файле; в Android используется его аналог под названием Toybox. Как и в «обычных» дистрибутивах Linux (и в отличие встраиваемых систем), основным способом взаимодействия с системой является графический интерфейс, а не командная строка. не менее, «добраться» до командной строки очень просто — достаточно запустить приложение-эмулятор терминала. По умолчан обычно не установлен, но его легко, например, скачать из Play Store (Terminal Emulator for Android, Material Terminal, Termux). Во мн «продвинутых» дистрибутивах Android — таких, как LineageOS (бывший CyanogenMod) — эмулятор терминала предустановлен.

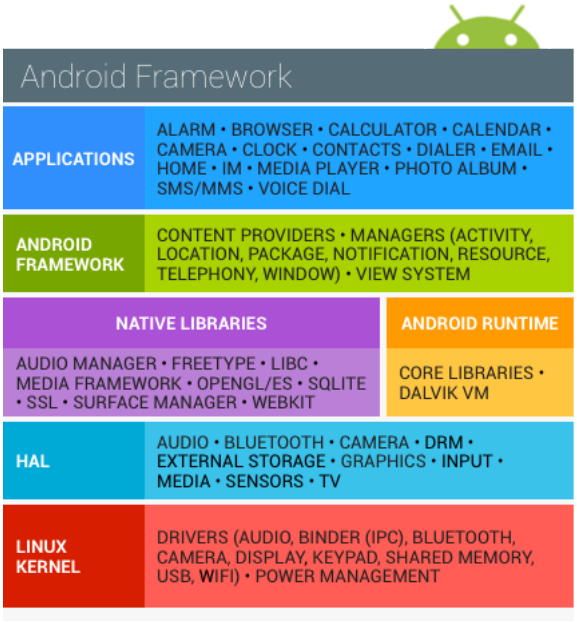


Второй вариант — подключиться к Android-устройству с компьютера через Android Debug Bridge (adb). Это очень похоже на подключение через SSH:

```
user@desktop-linux$ adb shell
android$ uname
Linux
```

Из других знакомых компонентов в Android используются библиотека FreeType (для отображения текста), графические API OpenGL EGL и Vulkan, а также легковесная СУБД SQLite.

Кроме того, раньше для реализации WebView использовался браузерный движок WebKit, но начиная с версии 7.0 вместо этого используется установленное приложение Chrome (или другое; список приложений, которым разрешено выступать в качестве WebV provider, конфигурируется на этапе компиляции системы). Внутри себя Chrome тоже использует основанный на WebKit движок Blink отличие от системной библиотеки, Chrome обновляется через Play Store — таким образом, все приложения, использующие WebVie автоматически получают последние улучшения и исправления уязвимостей.



It’s all about apps

Как легко заметить, использование Android принципиально отличается от использования «обычного Linux» — вам не нужно открыв закрывать приложения, вы просто переключаетесь между ними, как будто все приложения запущены всегда. Действительно, одна уникальных особенностей Android — в том, что приложения не контролируют напрямую процесс, в котором они запущены. Давайте поговорим об этом подробнее.

Основная единица в Unix-подобных системах — процесс. И низкоуровневые системные сервисы, и отдельные команды в shell’е, и графические приложения — это процессы. В большинстве случаев процесс представляет собой чёрный ящик для остальной систе другие компоненты системы не знают и не заботятся о его состоянии. Процесс начинает выполняться с вызова функции `main()` (на самом деле `_start`), и дальше реализует какую-то свою логику, взаимодействуя с остальной системой через системные вызовы и простейшее межпроцессное общение (IPC).

Поскольку Android тоже Unix-подобен, всё это верно и для него, но в то время как низкоуровневые части — на уровне Unix — опер понятием процесса, на более высоком уровне — уровне Android Framework — основной единицей является *приложение*. Приложен не чёрный ящик: оно состоит из отдельных компонентов, хорошо известных остальной системе.

У приложений Android нет функции `main()`, нет одной точки входа. Вообще, Android максимально абстрагирует понятие *приложени запущено* как от пользователя, так и от разработчика. Конечно, процесс приложения нужно запускать и останавливать, но Android делает это автоматически (подробнее я расскажу об этом в следующих статьях). Разработчику предлагается реализовать несколь отдельных компонентов, каждый из который обладает своим собственным жизненным циклом.

> In Android, however, we explicitly decided we were not going to have a main() function, because we needed to give the platform more control over how an app runs. In particular, we wanted to build a system where the user never needed to think about starting and stoppin apps, but rather the system took care of this for them… so the system had to have some more information about what is going on inside each app, and be able to launch apps in various well-defined ways whenever it is needed even if it currently isn’t running.

Для реализации такой системы нужно, чтобы приложения имели возможность общатся друг с другом и с системными сервисами – другими словами, нужен очень продвинутый и быстрый механизм IPC.

Этот механизм — Binder.

Binder

Binder — это платформа для быстрого, удобного и объектно-ориентированного межпроцессного взаимодействия.

Разработка Binder началась в Be Inc. (для BeOS), затем он был портирован на Linux и открыт. Основной разработчик Binder, Dianne Hackborn, была и остаётся одним из основных разработчиков Android. За время разработки Android Binder был полностью перепис

Binder работает не поверх System V IPC (которое даже не поддерживается в bionic), а использует свой небольшой модуль ядра, взаимодействие с которым из userspace происходит через системные вызовы (в основном `ioctl`) на «виртуальном устройстве»

`/dev/binder`. Со стороны userspace низкоуровневая работа с Binder, в том числе взаимодействие с `/dev/binder` и marshalling/unmars данных, реализована в библиотеке libbinder.

Низкоуровневые части Binder оперируют в терминах объектов, которые могут пересылаться между процессами. При этом использ подсчёт ссылок (reference-counting) для автоматического освобождения неиспользуемых общих ресурсов и уведомление о заверш удалённого процесса (link-to-death) для освобождения ресурсов внутри процесса.

Высокоуровневые части Binder работают в терминах интерфейсов, сервисов и прокси-объектов. Описание интерфейса, предоставляемого программой другим программам, записывается на специальном языке AIDL (Android Interface Definition Languag внешне очень похожем на объявление интерфейсов в Java. По этому описанию автоматически генерируется настоящий Java-интер который потом может использоваться и клиентами, и самим сервисом. Кроме того, по `.aidl`-файлу автоматически генерируются д специальных класса: Proxy (для использования со стороны клиента) и Stub (со стороны сервиса), реализующие этот интерфейс.

Для Java-кода в процессе-клиенте прокси-объект выглядит как обычный Java-объект, который реализует наш интерфейс, и этот ко может просто вызывать его методы. При этом сгенерированная реализация прокси-объекта автоматически сериализует переданны аргументы, общается с процессом-сервисом через libbinder, десериализует переданный назад результат вызова и возвращает его Java-метода.

Stub работает наоборот: он принимает входящие вызовы через libbinder, десериализует аргументы, вызывает абстрактную реализа метода, сериализует возвращаемое значение и передаёт его процессу-клиенту. Соответственно, для реализации сервиса програм достаточно реализовать абстрактные методы в унаследованном от Stub классе.

Такая реализация Binder на уровне Java позволяет большинству кода использовать прокси-объект, вообще не задумываясь о том, ч функциональность реализована в другом процессе. Для обеспечения полной прозрачности Binder поддерживает вложенные и рекурсивные межпроцессные вызовы. Более того, использование Binder со стороны клиента выглядит совершенно одинаково, независимо от того, расположена ли реализация используемого сервиса в том же или в отдельном процессе.

Для того, чтобы разные процессы могли «найти» сервисы друг друга, в Android есть специальный сервис ServiceManager, который хранит, регистрирует и выдаёт токены всех остальных сервисов.

Binder широко используется в Android для реализации системных сервисов (например, пакетного менеджера и буфера обмена), но детали этого скрыты от разработчика приложений высокоуровневыми классами в Android Framework, такими как Activity, Intent и C Приложения могут также использовать Binder для предоставления друг другу собственных сервисов — например, приложение Goo Play Services вообще не имеет собственного графического интерфейса для пользователя, но предоставляет разработчикам других приложений возможность пользоваться сервисами Google Play.

Подробнее про Binder можно узнать по этим ссылкам:

- Android Binder—Embedded Linux Wiki
- Android Interface Definition Language, IBinder
- Deep Dive into Android IPC/Binder Framework
- Android Binder—Android Interprocess Communication
- An Overview of Android Binder Framework
- Binders & Window Tokens

В следующей статье я расскажу о некоторых идеях, на которых построены высокоуровневые части Android, о нескольких его предшественниках и о базовых механизмах обеспечения безопасности.

Метки: android internals, android, linux, binder

Facebook

Поделиться публикацией

ПОХОЖИЕ ПУБЛИКАЦИИ

20 сентября в 14:15

Как работает Android, часть 2

+71 34,4k 310 34

Комментарии 49

esata 03.08.17 в 15:51

Хорошая статья, также рекомендую почитать интересующимся книгу Embedded Android.
P.S — В Android используется Toolbox а не Toybox.

> **bugaevc** 03.08.17 в 15:56
>
> Раньше использовался Toolbox, с версии 6.0 Marshmallow перешли на Toybox. Подробнее можно почитать здесь: на LWN и в issue tracker'е

Mikhail_dev 04.08.17 в 05:07

Embedded Android вышла в 2013 году и базируется на андроиде 4.0, на сколько я помню. Хоть много чего не поменялось, но мне всё же кажется, что книга морально устарела. К примеру Security часть.

> **sigizmund** 04.08.17 в 12:53
>
> К сожалению это факт, отличная книга но сильно устарела. Вроде бы хотели выпустить новое издание, я на него даже подписался на Амазоне, но публикацию почему-то отменили.

Amareis 03.08.17 в 21:18

> Dianne Hackborn

С такой фамилией у неё, кажется, просто не было выбора. Разве что лесорубом стать :)

Shchvova 04.08.17 в 00:44

Программирую под NDK с самого зарождения этой поделки. Я не знаю кто вам сказал что bionic значительно быстрее, меньше и менее требовательна к памяти чем что либо другое — наврал. Сильно. По моим тестам bionic примерно в 5 раз медленнее. А о совместимости хот с ANSI C и говорить не стоит. Мои впечатления хорошо описывает бывший коллега в начале этого доклада https://academy.realm.io/posts/swi on-android/.

> **bugaevc** 04.08.17 в 01:30
>
> По крайней мере, так утверждают сами Google (видео, слайды). Есть ещё другая известная «лёгкая» libc — musl, у неё гораздо лучше со стандартами, и, возможно, musl и bionic в большой степени объединятся (видео, слайды).

Mikhail_dev 04.08.17 в 07:06

Спасибо за статью. Пишите еще по этой теме. Статей по фреймворку мало, а книг так и вообще по пальцам одной руки можно пересчитать, этом все они вышли достаточно давно.

Areso 04.08.17 в 07:42

> Как легко заметить, использование Android принципиально отличается от использования «обычного Linux» — вам не нужно открывать и закрывать приложения, вы просто переключаетесь между ними, как будто все приложения запущены всегда. Действительно, одна из уника особенностей Android — в том, что приложения не контролируют напрямую процесс, в котором они запущены. Давайте поговорим об этом подробнее
>
> ...
>
> У приложений Android нет функции main(), нет одной точки входа. Вообще, Android максимально абстрагирует понятие приложение запущ как от пользователя, так и от разработчика. Конечно, процесс приложения нужно запускать и останавливать, но Android делает это автоматически (подробнее я расскажу об этом в следующих статьях). Разработчику предлагается реализовать несколько отдельных компон

каждая из который обладает своим собственным жизненным циклом.

В результате никогда не знаешь, закрылось приложение, пока ты лазил в словарик, или нет. Невероятно бесит и плохо влияет на продуктив работы.

НЕКОТ 04.08.17 в 10:21

Присоединяюсь. Одно из проявлений парадигмы «наше железо/софт умнее тупого юзера».

Mikhail_dev 04.08.17 в 12:10

Быстрее не железо умнее, а софт. И таки да, это вполне нормально. IOS вообще весь закрытый со всех сторон.

НЕКОТ 04.08.17 в 12:22

Это я написал в связке «железо/софт». Понятно, что решения принимает софт, но часто это выглядит как железо. Например, АКПП машине — «железо», хотя идиотский алгоритм управления в ней — софт (фирмварий).

Но самое грустное — когда к этому «умному» железо-софту приделывают костыли, чтобы юзер мог этой фигнёй пользоваться. Возвращаясь к АКПП, можно вспомнить гениальнейшую кнопку «Overdrive off» в дополнение к позициям D-3-2 на коробке Тойоты Хайлендер. Форд Фалькон того же времени выпуска, к примеру, имеет вполне чёткие два режима «само переключается» и «водите переключает, а мы только следим, чтобы движок не перекрутился и не заглох».

novice2001 04.08.17 в 14:53

И в чем проблема с кнопкой?

НЕКОТ 04.08.17 в 15:27

Костыль это. Кривой.

novice2001 04.08.17 в 15:36

В чем костыльность-то? Для меня, как пользователя аналогичной АКПП, это совершенно неочевидно.

НЕКОТ 04.08.17 в 15:57

Ну во-первых, увеличение количества ненеобходимых сущностей. Вместо одной степени свободы даётся как бы две (поз селектора и кнопочка).

Во-вторых, если после дефолтного значения нажать кнопку, и лампочка загорается, то чисто интуитивно это значит, что что-то включили. А на самом деле выключается возможность включения овердрайва. Я не раз слышал «Включи овердра Т.е. люди путаются.

В-третьих, нет индикации выбранной передачи. Есть индикация выбранного режима в двух местах: положение селектора лампа «овердрайв выключен».

В-четвёртых, в мануалке написано (да, я читал), что «3» и «2» можно включать только на спуске.

В итоге непонятно, а что вообще происходит. С учётом странного алгоритма автомата, происходит полная дизориентаци юзера. Как я и писал выше, «наше железо/софт умнее тупого юзера».

Для сравнения на Форде либо едем на автомате, либо сами решаем, когда и куда переключать. Надо на подъёме с прице обогнать — не вопрос. Заранее ручку в сторону толкнул и перед обгоном понизил. Дальше только тапку в пол. Всё ~~тепло ламповое~~ просто и понятно.

novice2001 07.08.17 в 23:23

1. Степеней свободы там действительно скорее всего две.
Во всяком случае это верно для Mitsubishi. O/D off не просто ограничивает переключение вверх как селектор АКПП, нс меняет алгоритм переключения.

2. Нет, это не означает именно включение, это означает отклонение от некоего дефолтного поведения. Сбойнул датчи **отключилась** АБС — загорелась лампа, **отключили** систему стабилизации — загорелась лампа. То, что люди говорят говорит лишь об их безграмотности, не более того.

3. А она точно нужна — эта индикация?

4. А включение этих режимов и имеет смысл только на спуске для торможения двигателем. Во всяком случае я не мог придумать для чего еще их имеет смысл использовать.

Не происходит никакой дезориентации. Подавляющему большинству режимы кроме D просто не нужны.Если кто-то зн что точно ему нужно, то изучить логику управления из целого одного рычага и целой одной кнопки — дело 5 минут.

По вашей логике владельцы полноценных внедорожников и вовсе не должны с места трогаться из-за дезориентации – куда больше рычагов и/или кнопок. Про самолеты с целыми панелями выключателей и индикаторов вовсе молчу.

НЕКОТ 08.08.17 в 03:10

Нет, по моей логике логика должна быть простая. У владельцев «полноценных внедорожников» логика минимальна их количества рычагов.

«изучить логику управления из целого одного рычага и целой одной кнопки» — я за этим в мануалку и полез. Но по мнению Тойоты, понимать логику — не дело юзера. Они её там не описывают.

9660 12.08.17 в 13:51

Сколько видел автоматов, везде логика одна и та же. Езда с выключенным овердрайвом считается ненормальной загорается транспарант.
На вполне полноценных внедорожниках в том числе.
Кстати, абсолютно аналогично обстоит дело с раздаткой, когда выключены обе оси, загорается лампа, когда включена одна из осей лампа тухнет.

Мне подобный подход объясняли авиаторы. Любой загоревшийся транспарант это сигнал отхода от нормальной ситуации.

НЕКОТ 08.08.17 в 03:11

Да, предлагаю остаться при своих. Очевидно, существует некоторе ненулевое множетво юзеров, довольных логико Тойоты и логикой Форда.

Вы в первом множестве, я во втором.

jonic 04.08.17 в 14:03

При этом ios с гигом памяти не убивает последние приложения моментально. Это действительно бесит когда вышел в вк ответить, ютуб закрылся на середине просмотра видео. (у меня 2 телефона, android для разработки и youtube из за экрана большего размера Недавно проверял, кстати, для друга. ios выдержал в памяти gta vicecity, badland2, youtube, vk, whatsapp, safari — при этом я писал в с экрана. Но тут мне кажется всё таки от вендора зависит больше, как агрессивно будет работать сборщик мусора

Pro-invader 04.08.17 в 15:16

Не сборщик мусора, а Lowmemorykiller.

bugaevc 04.08.17 в 11:13

Я подробнее расскажу об этом в одной из следующих статей (хотя что здесь рассказывать — `savedInstanceState`), но идея в том, что вы должны замечать автоматическое закрытие приложения, потому что оно откроется ровно в том же месте и в таком же состоянии (скролл введённый текст и т.п.), в котором вы его оставили.

От настоящего переключения это отличается в основном чуть большим временем, которое нужно на переключение (приложению нужно запуститься). Сравните с выгрузкой программы в swap — она делается по тем же причинам (нехватка памяти) и при переключении на выбранную программу точно так же требуется дополнительное время, чтобы восстановить программу в таком же состоянии, как вы её оставили.

Если какое-то приложение работает не так (а, например, открывается на стартовой странице), то это неправильно написанное приложен

basnopisets 04.08.17 в 12:17

> Если какое-то приложение работает не так (а, например, открывается на стартовой странице), то это неправильно написанное приложение.

Не всегда. Есть кейсы, когда возобновлять приложение на прежней стадии не нужно, а важнее отобразить актуальные данные. Вообще этот вопрос из области UX и каждый UX-дизайнер и PO дает на него свой собственный ответ в зависимости от своего случая

bugaevc 04.08.17 в 12:48

Отображать актуальные данные нужно всегда, независимо от перезапуска приложения. Но да, в этом плане Android очень гибок и позволяет настраивать такие параметры, как `alwaysRetainTaskState` (по умолчанию состояние через некоторое время сбрасывается) и `clearTaskOnLaunch`, то есть приложение может выбрать оптимальный для своей специфики режим работы.

Но если в результате

> Невероятно бесит и плохо влияет на продуктивность работы

то это определённо неправильно сделанное приложение.

Jogger 04.08.17 в 20:09

Боюсь это означает, что подавляющее большинство приложений в гугл плей написаны неправильно. А это в свою очередь означает, чт была выбрана неверная парадигма — то есть, ключевой функционал переложили на плечи прикладных программистов, которые, как и следовало ожидать, забили на это болт. Собственно, это одна из причин, по которым андроид defective by design. Иногда к нему приделывают костыли (возможность «закрепить» приложение, запретить его закрывать), которые немного улучшают ситуацию, но это полумеры.

bugaevc 04.08.17 в 21:03

> ключевой функционал переложили на плечи прикладных программистов, которые, как и следовало ожидать, забили на это болт

О каком функционале вы говорите? То, что я назвал — восстановление activity, введённого текста, состояния переключателей, скро т.п. — делается *автоматически* и *по умолчанию* Android Framewok'ом. Кроме того, разработчику приложения даётся возможность эффективно сохранить и потом восстановить кастомную информацию.

Я бы не назвал это defective by design, совсем нет.

> Иногда к нему приделывают костыли (возможность «закрепить» приложение, запретить его закрывать)...

Если вы про screen pinning, то это *вообще не для того*. Это фича, которая позволяет запретить переключение между приложениями снятия запрета требует пароль/код). Используется, например, чтобы ребёнок делал на вашем устройстве только то, что вы ему разрешили, или в embedded Android (разного сорта банкоматы, терминалы...); к app lifecycle это не имеет никакого отношения.

> которые немного улучшают ситуацию, но это полумеры.

Посмотрите новые Android Architecture Components, это далеко не полумеры.

Jogger 04.08.17 в 21:47

Я в данном случае выступаю как пользователь, а не как программист, и соответственно пишу о личном опыте. А опыт таков (и суд другим комментариям — я далеко не уникален), что большинство пользовательских программ на андроид склонны к спонтанном забыванию текущих данных, то есть соответствуют тому, что вы называете «неправильно написанное приложение.». Ещё раз — подавляющее большинство. Если большинство программистов пишет неправильные программы — это явный проёб в парадигме системы, а как по мне это именно то, что называют «defective by design». Даже странно что вы это отрицаете — у вас что, никогд было смартфона на android?

> Если вы про screen pinning

Нет, совершенно не про это. Просто встречал как-то прошивку, где вместо списка последних запущенных программ — список програм, находящихся в памяти. И соответственно выбрасывание из списка выгружало программу, а закрепление в списке — не позволяло её выгружать (я не знаю, как бы повёл себя Lowmemorykiller если бы я попытался закрепить слишком много программ) такая система была на порядок удобнее, чем то, что я встречаю в большинстве прошивок.

Goodkat 21.09.17 в 23:55

На конкурирующей платформе премодерация приложений немного помогает и отфильтровывает откровенный шлак, но и там, вкладка браузера может закрыться с неотправленным комментарием, если вдруг не хватит памяти на входящий звонок. Думаю, проблему бы решил своп — выгрузка/загрузка памяти приложения целиком, а не сохранение состояния руками программиста.

edo1h 06.08.17 в 20:05

сразу вспоминается palmos (версий до 4.x включительно), система однозадачная, но приложения (по замыслу заработчиков os) открываются моментально и на том же месте, где ты их оставил.

saboteur_kiev 06.08.17 в 21:35

В WebOS что-то переделали разве?

Mikhail_dev 04.08.17 в 12:04

Вот как раз таки нормально влияет на продуктивность работы, в лучшую сторону. Если ОЗУ хватает, то и нечего выгружать приложения. Е различный уровень у процессов, которые при нехватке памяти выгружаются фреймворком. Их можно глянуть через «adb shell dumpsys meminfo»

Areso 04.08.17 в 12:17

Мой опыт показывает ровно обратное. Часть ненужных вещей в фоне крутится до тех пор, пока не тапшешь по виджету очистки памяти часть выгружается при любом удобном поводе, хотя свободной памяти еще более, чем достаточно.
У меня сейчас второй смарт и второй компьютер обладают равным количеством ОЗУ, так вот, пользоваться компьютером как многозадачным устройством сильно удобнее.

Pro-invader 04.08.17 в 15:10

А вы на смарте пробовали SWAP включать. Я пробовал, ставил 3 Гб, ничего не выгружалось много дней, даже тяжелые приложения. в том, что не все ядра прошивок поддерживают SWAP. У самсунга, например, нет.

Goodkat 22.09.17 в 00:03

Загруженным из свопа файрфоксом пользоваться решительно невозможно. Проще прибить его процесс и запустить заново с загрузкой всех вкладок, чем дожидаться, пока он вылезает из свопа. Причём он не загружается весь сразу, а как-то кусками. Вро просвопился, всё работает, потом проскроллишь чуток или в соседнюю вкладку тыкнешь, и можно идти заваривать чай, пока он прочухается.

Но своп на смартфоне действительно помогал, правда я не пользовался им уже лет пять — 1 GB оперативки хватает :)

GloooM 04.08.17 в 10:07

А может быть тут подскажут, есть в кастомных прошивках на основе Nougat такой баг с некоторыми приложениями, в частности приложени Вконтакте, при переключении типа сети WiFi <-> 4G/3G теряет соединение с сетью и пишет «Соединение» пока не выгрузишь и не запусти заново. При этом другие приложения, не подверженные багу, работают исправно. Что примечательно недавно вышел релиз ParanoidAndroid там этот баг не проявляется, интересно бы понять в какой подсистеме он может возникать и сравнить что сделали/не сделали в PA в сравне с LOS, где баг проявляется.

sigizmund 04.08.17 в 12:49

Очень рекомендую к прочтению последнее издание Танненбаума "Modern Operating Systems" где собственно сама Dianne Hackborn написа главу про Андроид (можно довольно легко найти PDF в гугле). Среди разработчиков Андроида эта глава считается одним из самых лучших и довольно кратких) введений в архитектуру Android.

basnopisets 29.09.17 в 11:51

еще недавно был доклад Эфи Барак

sigizmund 04.08.17 в 12:51

> например, приложение Google Play Services вообще не имеет собственного графического интерфейса для пользователя, но предоставляет разработчикам других приложений возможность пользоваться сервисами Google Play.

Это не совсем верно. Многие интерфейсы которые вы видите в Settings на самом деле реализованы в GmsCore. Например, немалая часть G Play Protect сделана именно там (оставшая часть интерфейса — в Play Store app).

bugaevc 04.08.17 в 13:06

Согласен, но у GPS нет именно *собственного* интерфейса, нет MainActivity, пользователь не видит это приложение в лончере (хотя в стар версиях Android оно отображалось как Google Settings, но теперь интегрировано в системные настройки).

Подозреваю, что и отображение Maps сделано внутри Play Services. Не подскажете, где почитать про это (и про то, как реализован интер Play Protect) подробнее?

sigizmund 04.08.17 в 18:33

Насчет Maps не уверен, но подозреваю что как минимум часть отображения сделана именно там. UI Play Protect реализован как activit GmsCore (Play Services), которая каким-то хитрым образом в Android ОС цепляется к System Settings (честно говоря, не разбирался ка именно) + часть в Play Store.

moden 04.08.17 в 21:45

Почему процессы, созданные через терминал прибиваются через время? Запускал в скрине минимальный сервер на Python aiohttp, работа минут 20. Нагрузки, конечно никакой.
Если это ожидаемое поведение, есть способы «договориться»? Сервисом оформить или по-другому...

bugaevc 04.08.17 в 21:48

Да, нужно оформлять foreground service-ом. Подробнее про это — в следующей статье.

bugaevc 21.09.17 в 11:37

UPD: про это будет в третьей статье.

Alex42rus 08.08.17 в 10:22

Расскажите почему в андроид до сих пор не исправили лаги в интерфейсе, а также почему общая скорость работы падает со временем.

gibson_dev 21.09.17 в 11:16

Все таки не

> нескольких компонент

а

> нескольких компонентов

Компоне́нта — составляющая чего-либо; термин, используемый в математике и физике; сфера употребления термина компонента у́же, чем слова компонент, и включает в себя только терминологические контексты

bugaevc 21.09.17 в 11:32

Спасибо, вы совершенно правы. Исправил в статье.

Lsh 28.09.17 в 13:53

Кое-что не понятно в том, как Binder работает на низком уровне. Приложения его открывают как файл и некоторые данные из него отображаются в память (mmap), так? Когда нужно выполнить метод «с другой стороны», то вызывающее приложение делает ioctl, так? Как отвечающее приложение об этом узнаёт? Оно ждёт в каком-то блокирующем вызове или через select?

bugaevc 28.09.17 в 20:30

Да, ждёт в блокирующем вызове `ioctl(binder_fd, BINDER_WRITE_READ, &bwd)` — он работает примерно так же, как `write()` и `read()` по на этом дескрипторе. Вот реализация этого ioctl со стороны ядра.

Да, `binder_fd` поддерживает и poll, но обычно процессы блокируют свой основной поток (он же looper thread, он же UI thread в приложен на `binder_write_read` и делают остальное блокирующее I/O (например, сетевые запросы) в других потоках.

Только полноправные пользователи могут оставлять комментарии. Войдите, пожалуйста.

На пути к естественному интеллекту

+11 1,4k 19 4

Хаскель — ход конем II GT

+9 1,3k 6 4

Простая Scada на Python и Arduino

+11 2,7k 29 4

Sportiduino. Система электронной отметки для спортивного ориентирования. Часть 2 GT

+18 1,7k 16 0