1. Понятие встраиваемой системы, операционной системы, процессорной архитектуры.

Встра́иваемая систе́ма (встро́енная систе́ма, англ. embedded system) — специализированная микропроцессорная система управления, контроля и мониторинга.
Особенности встраиваемых систем:

- Минимальное энергопотребление.
- Минимальные габариты и вес.
- Защита минимальна и обеспечивается прочностью и жесткостью конструкции.
- Обеспечение минимума требований тепловых режимов.
- Микропроцессор, системная логика, ключевые микросхемы на одном кристалле.
- Специальные требования с учетом области применения.

Основа встроенных систем:

- Одноплатные (однокристальные) ЭВМ.
- Специализированные микропроцессоры.
- Программи́руемая логи́ческая интегра́льная схе́ма (ПЛИС)

Операцио́нная систе́ма — комплекс взаимосвязанных программ, предназначенных для управления ресурсами компьютера и организации взаимодействия с пользователем.
Операционные системы нужны:

- если нужен универсальный механизм сохранения данных;
- для предоставления системным библиотекам часто используемых подпрограмм;
- для распределения полномочий;
- необходима возможность имитации «одновременного» исполнения нескольких программ на одном компьютере;
- для управления процессами выполнения отдельных программ

Классификация ОС по типу ядра:

- монолитные системы,
- ОС с монолитным ядром,
- микроядерные ОС,
- гибридные системы.

С программной точки зрения архитектура процессора — это совместимость с определённым набором команд (Intel x86), их структуры (система адресации, набор регистров) и способа исполнения (счётчик команд).

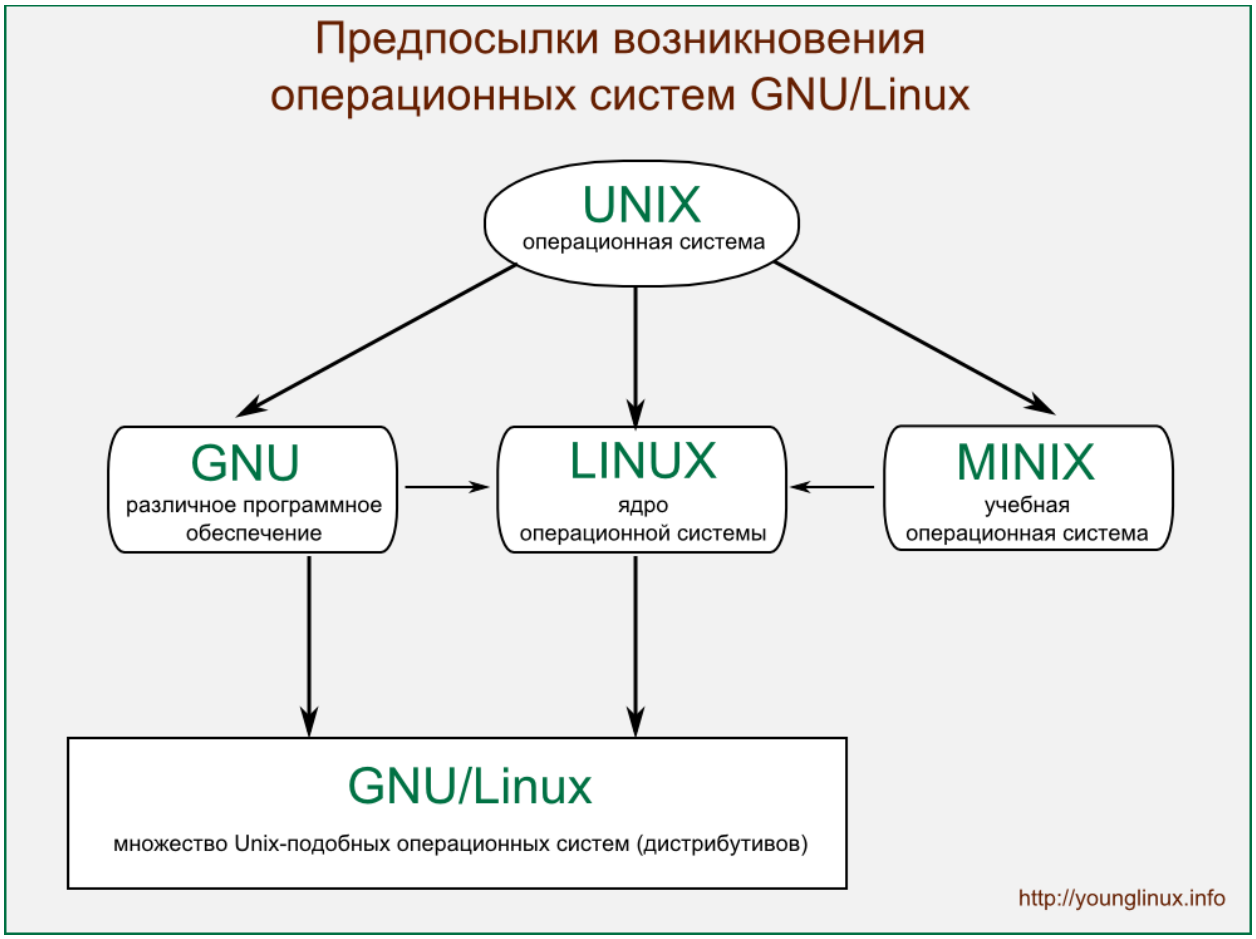
С аппаратной точки зрения архитектура процессора — это некий набор свойств и качеств, присущий целому семейству процессоров (Skylake – процессоры Intel Core 5 и 6 поколений).

Выделяют следующие основные архитектуры процессора:

- RISC
- CISC
- MISC
- VLIM

2. Предпосылки возникновения и история операционных систем ОС Linux.

Операционные системы на базе ядра Linux представляют собой большую группу Unix-подобных ОС. Конкретную разновидность принято называть дистрибутивом Linux. Появление систем GNU/Linux можно считать естественной эволюцией компьютерных технологий.



Linux или GNU/Linux — общее название UNIX-подобных операционных систем на основе свободного ядра Linux и собранных для него библиотек и системных программ, разработанных в рамках проекта GNU.

Истоки UNIX

- 60-е годы 20 в. — государственные и военные структуры США вкладывают средства в наукоемкие проекты: разработку новых вычислительных систем, средств хранения и передачи данных
- 60-е годы — Multics (Bell Labs / AT&T)
- 1969 — первая версия UNIX для PDP-7 (Кен Томпсон (Ken Thompson) и Деннис Ричи (Dennis Ritchie))
- 1972 — версии UNIX для PDP-11 (написана на C)
- 70-е годы — версия UNIX от университета в Беркли (University of California-Berkeley)
- 1977 г. — первая версия Berkeley Software Distribution
- После этого началось развитие UNIX в двух направлениях: одни команды разработчиков взяли за основу версию Беркли, другие — версию System V Release 4.2, которая была разработана AT&T.

GNU - (рекурсивный акроним от англ. *GNU's Not UNIX* — «GNU не UNIX») — свободная Unix-подобная операционная система, разрабатываемая Проектом GNU.

Ричард Мэттью Столлман - Основатель движения свободного ПО, проекта GNU, Фонда свободных программ и Лиги за свободу программирования.

Линус Бенедикт Торвальдс

- Родился 28 декабря 1969 в Хельсинки — финский программист, хакер.
- Создал Linux — ядро операционной системы GNU/Linux, являющейся на данный момент самой распространённой свободной операционной системой.

Свойства операционной системы Linux

- Многопользовательский режим
- Многозадачность
- Многоплатформенность
- Способность к взаимодействию
- Масштабируемость
- Переносимость
- Гибкость
- Надежность

- 1994 — первая официальная версия Linux
- 1995 — регистрация товарного знака Linux
- 1996 — Выбран символ Linux — пингвин Tux
- 1996 — Linux 2.0.0
- 1999 — Linux 2.2.0
- 2001 — Linux 2.4.0
- 2003 — Linux 2.6.0
- 2011 — Linux 3.0
- 2012 — Linux 3.3

- 2015 — Linux 4.0
- 2018 — Linux 4.15

Дистрибутив (от англ. distribute распространять) — это форма распространения программного (часто — системного) обеспечения.
Дистрибутив обычно содержит программы для начальной инициализации системы (в случае дистрибутива операционной системы — инициализация аппаратной части, загрузка урезанной версии системы и запуск программы установщика), программу установщик (для выбора режимов и параметров установки) и набор специальных файлов, содержащих отдельные части системы (пакеты).

История BSD

▪1974 – исследования Unix в университете в Беркли.
▪1977 – первая BSD-версия создана аспирантом Билли Джоем (Bill Joy).
▪1978, начало года – выпущена первая дистрибутивная версия программ Беркли (Berkeley Software Distribution - BSD).
▪1978, конец года – выпущен второй дистрибутив 2BSD.
▪1979 – 3BSD
▪1980 – Университет в Беркли становится центром разработок BSD-версии.
▪1980 – 4BSD.
▪1980 – заказ от DARPA на разработку TCP/IP.
▪1981 – 4.1.BSD.
▪1983, сентябрь – 4.2BSD.
▪1993 – 4.4BSD.
▪1990 – серия статей о переносе BSD Unix на ПК с процессором 386-серии.
▪1992 – выпуск BSD для PC (386).

BSD системы

▪FreeBSD (FreeBSD - это зарегистрированная торговая марка FreeBSD Foundation.):
  ▫PicoBSD
  ▫Frenzy
  ▫PC-BSD
▪NetBSD
▪OpenBSD
▪BSDI
▪DragonFlyBSD

***Цель** проекта FreeBSD - предоставление программного обеспечения, которое может быть использовано для любых целей и без дополнительных ограничений.*

Наиболее популярными дистрибутивами являются (расположены в алфавитном порядке названия их пакетных форматов): deb-based (Debian, Mint, Ubuntu), pacman-based (Arch Linux, Chakra, Manjaro), RPM-based (RedHat, Fedora, Mageia, OpenSUSE), source-based (Slackware, Gentoo).

- Debian — операционная система, состоящая как из свободного ПО с открытым

исходным кодом, так и из закрытых компонентов. В первичной форме, Debian GNU/Linux — один из самых популярных дистрибутивов Linux, оказывающий значительное влияние на развитие этого типа ОС в целом.

- Ubuntu — дистрибутив операционной системы Linux, основанный на Debian GNU/Linux. Проект активно развивается и поддерживается свободным сообществом.
- Red Hat — американская компания, выпускающая решения на основе свободной операционной системы GNU/Linux: Red Hat Enterprise Linux (распространяется по годовой подписке) и Fedora (распространяется свободно), а также другие программные продукты и услуги на основе открытого исходного кода.
- openSUSE — дистрибутив Linux. Изначально разрабатывался в Германии, но сейчас его владельцем является американская корпорация Novell, Inc. Дистрибутив широко используется во всем мире, особенно в Германии. Был основан на дистрибутиве Slackware, однако был значительно переделан и представляет собой обособленный дистрибутив, отличается от последнего форматом пакетов, а также системой настройки и администрирования YaST. Со временем SUSE включила в себя много аспектов Red Hat Linux (использование системы RPM и /etc/sysconfig). Цикл выпуска новых версий — 1 год.

GNU/Linux - основные дистрибутивы

- Gentoo —популярный дистрибутив GNU/Linux с мощной и гибкой технологией Portage, совмещающей в себе возможности конфигурирования и настройки, а также автоматизированную систему управления пакетами. Последняя создавалась под влиянием системы управления пакетами во FreeBSD. Отличительной особенностью Gentoo является наличие оптимизации под конкретное аппаратное обеспечение.
- Knoppix — проект «Linux с диска»
- ASPLinux — российский, совместим с Red Hat
- ALT Linux— российский, упор на защищенность и надежность
- Scientific Linux — переработка Red Hat Enterprise Linux для научных работников силами CERN и Fermilab
- ArchLinux — rolling release

Интерфейсы пользователя

Графический интерфейс предназначен для массового пользователя, уровень управления заданиями;

Командный интерфейс позволяет использовать возможности ОС и ВС более гибко, предназначен для разработки прикладных программ и вычислительных заданий, управления вычислительными работами на уровне задач, данных и устройств

Графическая среда рабочего стола

- Gnome:
  - Gnome 2: Cinnamon, Mate
  - Gnome Shell

- KDE - KDE Plasma
- Enlightenment
- XFCE
- LXDE
- Unity

GNOME — свободная среда рабочего стола для Unix-подобных операционных систем. GNOME является частью проекта GNU.
Разработчики GNOME ориентируются на создание полностью свободной среды, доступной всем пользователям вне зависимости от их уровня технических навыков, физических ограничений и языка, на котором они говорят. В рамках проекта GNOME разрабатываются как приложения для конечных пользователей, так и набор инструментов для создания новых приложений, тесно интегрируемых в рабочую среду.
GNOME является дружественной к пользователю графической оболочкой, позволяющей пользователям легко использовать и настраивать свои компьютеры. В GNOME имеется панель (для запуска приложений и отображения их состояния), рабочий стол (где могут быть размещены данные и приложения), набор стандартных инструментов и приложений для рабочего стола, а также набор соглашений, облегчающих совместную работу и согласованность приложений.

KDE Software Compilation (KDE SC) — свободная среда рабочего стола и набор программ от проекта KDE. До начала 2010 года была известна как KDE (сокращение от K Desktop Environment). Построена на основе кросс-платформенного инструментария разработки пользовательского интерфейса Qt. Работает преимущественно под UNIX-подобными операционными системами, которые используют графические подсистемы X Window System и Wayland. Новое поколение технологии KDE 4 частично работает на Microsoft Windows и Mac OS X.

В состав KDE SC входит набор тесно интегрированных между собой программ для выполнения повседневной работы. Также в рамках проекта KDE разрабатываются интегрированная среда разработки KDevelop, офисный пакет Calligra Suite, музыкальный проигрыватель Amarok и многие другие. Эти программы не являются частью KDE SC.

- KDE является простой в использовании современной графической оболочкой. Вот лишь некоторые из преимуществ, которые даёт пользователю KDE:
- Прекрасный современный рабочий стол
- Рабочий стол, полностью прозрачный для работы в сети
- Интегрированная система помощи, обеспечивающая удобный и согласованный доступ к системе помощи по использованию рабочего стола KDE и его приложений.

3. История и тенденции развития мобильных операционных систем.

(в презенташках 1 слайд так что хз то ли)
Мобильная ОС -- операционная система для смартфонов, планшетов, КПК или других мобильных устройств. Мобильные операционные системы сочетают в себе

функциональность ОС для ПК с функциями для мобильных и карманных устройств: сенсорный экран, сотовая связь, Bluetooth, Wi-Fi, GPS-навигация, камера, видеокамера, распознавание речи, диктофон, музыкальный плеер, NFC и инфракрасное дистанционное управление.

Мобильные операционные системы: структура

- Android
- Bada
- Blackberry OS
- Firefox OS
- Open webOS
- iOS
- Linux - Maemo, MeeGo, Openmoko, Tizen, Sailfish OS, UbuntuPhone
- Palm OS
- Symbian OS
- Windows Mobile и Windows CE Windows Phone и Windows 10

(RIP картинка с долями рынка каждой из систем)

Symbian OS разработана и продвигается консорциумом Symbian. Консорциум Symbian был основан в июне 1998 года такими компаниями как: Nokia, Psion, Ericsson , Motorola. Symbian OS написан практически целиком на C++, но есть поддержка Java.

Главное достоинство: она предназначалась именно для мобильных устройств и их не сильно высоких технических мощностей;

Недостаток: Очень часто после выхода новой версии Symbian OS, смартфон не всегда совмещается со старыми программами;

24 января 2013 операционная система Symbian **переведена в режим поддержки**.

Android наиболее распространенная операционная система для мобильных устройств. Первая версия операционной системы вышла в 2008 году, после чего произошло несколько обновлений системы. ОС Android основана на ядре Linux.

Достоинства: Благодаря открытому исходному коду любой может создать свое приложение и альтернативные версии Андроид.

Недостатки:

- Множество актуальных версий - для многих устройств новая версия входит слишком поздно или не появляется вовсе, поэтому разработчикам приходится разрабатывать приложения, ориентируясь на более старые версии;
- Высокая предрасположенность к хакерским атакам из-за открытости кода;

Последняя версия 9.0 вышла 6 августа 2018

IOS - вторая по популярности мобильная операционная система, разрабатываемая Apple только для своих устройств. Впервые была представлена в 2007 году с мобильным телефоном iPhone, эта презентация показала направление развития для всех производителей смартфонов.

Достоинства:

- Защищенность - гаджет на платформе iOS сложно заразить вирусом или вывести из строя по незнанию
- Отсутствие программных сбоев - нет зависаний и странностей в поведении;
- · Высокая скорость работы;
- · Высокое качество софта и продуманность.

Недостатки:

- · Отсутствие пользовательских настроек;
- · Невозможность замены или удаления стандартных приложений;
- · Закрытость файловой системы - невозможность прямой переброски файлов в Apple iPhone, iPod и iPad, отсутствие возможностей для полного обзора содержимого устройства;

Последняя версия 12.1 вышла 30 октября 2018 года

1996 — Windows CE. Мобильная ОС с наиболее длинной историей, пережившая несколько реинкарнаций (Pocket PC, Windows Mobile, Windows Phone) и существующая по сей день. Кодовая база мобильных ОС Microsoft реализована на базе серверной Windows CE. Мобильная реализация Windows CE появилась примерно в середине 1990-х, но популярной стала после произведённого ребрендинга — Win CE версии 3.0 получила название Pocket PC 2000. Она была ориентирована в основном на технических энтузиастов и бизнес-сегмент, что не позволило ей в своё время проникнуть в массовый сегмент, как это удалось iOS и Android.

Windows Phone - мобильная операционная система, разработанная Microsoft, является преемником Windows Mobile.Windows Phone 7 - первая версия, была официально представлена 11 октября 2010 года

Достоинства:

- · Скорость и плавность работы интерфейса. Система не тормозит, она крайне отзывчива и весьма приятна в работе;
- · Нет проблемы нехватки оперативной памяти. Даже при небольшом объёме оперативной памяти, системой можно комфортно пользоваться. Если, конечно, не ставить «тяжёлых» игр и приложений;
- · По причине закрытости системы - малое количество вирусов;

Недостатки:

- · Скудный выбор приложений;
- · Абсолютно закрытая ОС, приложения только из MS Marketplace;
- · Медленное развитие платформы;

21 января 2015 года Microsoft официально представила следующее поколение мобильной ОС Windows 10 Mobile(Снова mobile вместо phone)

С 24 января 2018 года активная разработка прекращена

Blackberry OS - операционная система с основным набором приложений для смартфонов и коммуникаторов, выпускаемых компанией Research In Motion Limited (RIM).

Версии ОС:

- Первая версия - 19 января 1999
- BlackBerry OS 5.0 - выпущена компанией RIM в конце 2009 года;
- Последняя версия - BlackBerry OS 10 - 30 января 2013 года.

BlackBerry OS считается системой с высоким функционалом и стабильностью работы. Очень безопасная система(на это делается упор).

15 декабря 2017 объявили о конце разработки системы в течении 2ух ближайших лет.

Tizen - это операционная система с открытым исходным кодом, основанная на ядре Linux. ОС является наследником таких операционных систем как MeeGo, LiMo и bada. Изначально Tizen задумывалась как операционная система, разработка под которую должна была целиком происходить с использованием Web-технологий. Tizen поддерживается некоммерческим консорциумом развития Linux Foudation и рядом крупных компаний, входящих в консорциум Tizen Association(Intel, Samsung и др.)

Первая альфа-версия 5 января 2012

Последняя версия - Tizen 5.0 M2 (30 октября 2018)

Firefox OS -- это свободная ОС, с открытым исходным кодом. Используется ядро Linux, а интерфейс выполнен на HTML5. Ключевая идея, положенная в основу Firefox OS -- использование веб-движка Gecko и ориентация на веб-стандарты. Поддерживаются только те приложения, которые созданы на языках веб-программирования, т.е. HTML5, CSS и JavaScript.

Первая версия -. Firefox 1.0 TEF - 21 февраля 2013;

Последняя - 2.2.0 (29 апреля 2015 года)

В декабре 2015 года появилась информация о закрытии проекта как платформы для смартфонов

Ubuntu Phone OS (Ubuntu Touch) - мобильная платформа, разработанная компанией Canonical Ltd. для смартфонов и планшетов. ОС была анонсирована 2 января 2013 года и официально публично показана на выставке Consumer Electronics Show 8--11 января 2013 года.Ubuntu Phone базируется на настольной Desktop версии Ubuntu с заменой стандартной графической оболочки на мобильную версию Unity

Последняя версия - 14.09 RTM (2015)

5 апреля 2017 года прекращение разработки Ubuntu Touch

Sailfish OS - операционная система для смартфонов финской компании Jolla.В основе системы - ядро Linux, а интерфейс и приложения сделаны на QML и HTML5. Исходный код полностью открыт.

Первая версия 1.0.0.5 Kaajanlampi - 27 ноября 2013;

Последняя 2.2.0.29 7 июня 2018

Заключение: как видим существует множество различных мобильных операционных систем, однако большинство из них очень слабо распространены и в основном используются либо Android либо IOS

4. История и тенденции развития мобильной операционной системы Android.

В 2007 году Android Inc. , купленная Google в 2005 году, официально объявила о создании Open Handset Alliance (OHA) и анонсировала открытую мобильную платформу Android, а также в том же году альянс представил первую версию пакета для разработчиков Android «Early Look» SDK и эмулятор Android.

В 2008 году официально вышла первая версия операционной системы, а также первый полноценный пакет разработчика SDK 1.0, Release 1.

В 2009 году было представлено целых четыре обновления платформы. Так вышла версия 1.1 с исправлением различных ошибок. Ещё два обновления — 1.5 «Cupcake» и 1.6 «Donut». Обновление «Cupcake» привнесло существенные изменения: виртуальная клавиатура, воспроизведение и запись видео, браузер и другие. В «Donut» впервые

появились поддержка различных разрешений и плотности экрана и сетей CDMA. В октябре того же года вышла версия операционной системы Android 2.0 «Eclair» с поддержкой нескольких аккаунтов Google, поддержкой браузером языка HTML5 и других нововведений, а также после небольшого обновления в пределах версии «Eclair» появились «живые обои» и был видоизменён экран блокировки.

В середине 2010 года Google представила Android версии 2.2 под наименованием «Froyo», а в конце 2010 года — Android 2.3 «Gingerbread». После обновления «Froyo» стало возможно использовать смартфон в качестве точки доступа, использовать традиционную блокировку смартфона цифровым или буквенно-цифровым паролем, а обновление «Gingerbread» (Имбирный пряник) привнесло более полный контроль над функцией копирования и вставки, улучшение управления питанием и контроля над приложениями, поддержку нескольких камер на устройстве.

В 2011 году была официально представлена ориентированная на интернет-планшеты платформа Android 3.0 «Honeycomb».
Android 4.0 «Ice Cream Sandwich», вышедшая также в 2011 году, — первая универсальная платформа, которая предназначена как для планшетов, так и для смартфонов. Также обновление принесло новый интерфейс «Holo», который использовался до Android 4.4.4 KitKat .

В 2012 вышло обновление под названием «Jelly Bean» с порядковыми номерами 4.1, 4.2, 4.3.

В 2013 году Google представила следующую версию операционной системы Android 4.4, которая получила название шоколадного батончика «KitKat» по соглашению с компанией производителем Nestlé. Впервые KitKat появился на Nexus 5; эта версия Android оптимизирована для работы на более широком наборе устройств, имеющих 512 МБ ОЗУ в качестве рекомендуемого минимума. Также, в качестве тестовой опции, в настройках разработчика стала доступна виртуальная машина ART.

15 октября 2014 года была официально анонсирована Android 5.0 Lollipop. Главные обновления системы — новый дизайн Material Design и полный переход к виртуальной машине ART. Также, появился Project Volta, благодаря которому операционная система обращается к процессору не одиночными запросами, а пакетами данных, тем самым экономя заряд, в результате чего Nexus 5 может работать на 1,5 часа дольше.

9 декабря 2014 Google заменила официальную среду разработки, основанную на Eclipse (adt-bundle), на Android Studio.

В 2015 году была анонсирована операционная система для носимых устройств Android Wear(позже Wear OS). Также на Google IO были представлены версии Android Auto (для автомобилей) и Android TV (для телевизоров), тем самым Android перестал быть операционной системой только для мобильных устройств.
В 2015 Google представила Android 6 Marshmallow. Изменения: Android Pay, стандартизированная верификация при помощи отпечатков пальцев на уровне платформы, экономия заряда.

В 2016 году была представлена Android 7 Nougat. Изменения: Полноценная аппаратная поддержка режима виртуальной реальности, разделение экрана, очистка памяти, фильтрация звонков и т.д.

В 2017 году был представлен Android 8 Oreo. Изменения: просмотр видео в оконном режиме, динамические иконки, новые экранные жесты, бейджи на иконках, API нейронных сетей и общей памяти.

В 2018 году был представлен Android 9 Pie. Наряду с множеством “косметических” изменений была включена поддержка протокола безопасности DoT (DNS over TLS); также, для отправки запросов по протоколу HTTP, сервер, к которому данный запрос направляется, должен быть включен в специальный список, ссылка на который помещается в файл AndroidManifest.

5. Операционные системы реального времени.

Операционные системы реального времени (ОСРВ) – это операционные системы, способные обеспечить предсказуемое время обработки непредсказуемо возникающих внешних событий. Разделяют ОС «жесткого» и «мягкого» реального времени.

Операционная система, которая может обеспечить требуемое время выполнения задачи реального времени даже в худших случаях, называется *операционной системой жесткого реального времени*. Система, которая может обеспечить требуемое время выполнения задачи реального времени в среднем, называется *операционной системой мягкого реального времени*.

Системы жёсткого реального времени не допускают задержек реакции системы, так как это может привести к потере актуальности результатов, большим финансовым потерям или даже авариям и катастрофам. Ситуация, в которой обработка событий происходит за время, большее предусмотренного, в системе жёсткого реального времени считается фатальной ошибкой. При возникновении такой ситуации операционная система прерывает операцию и блокирует её, чтобы, насколько возможно, не пострадала надёжность и готовность остальной части системы. Примерами систем жёсткого

реального времени могут быть бортовые системы управления системы аварийной защиты, регистраторы аварийных событий.

В системе мягкого реального времени задержка реакции считается восстановимой ошибкой, которая может привести к увеличению стоимости результатов и снижению производительности, но не является фатальной. Примером может служить работа компьютерной сети. Если система не успела обработать очередной принятый пакет, это приведёт к остановке на передающей стороне и повторной посылке. Данные при этом не теряются, но производительность сети снижается.

Основное отличие систем жёсткого и мягкого реального времени можно охарактеризовать так: система жёсткого реального времени никогда не опоздает с реакцией на событие, система мягкого реального времени не должна опаздывать с реакцией на событие.

ОСОБЕННОСТИ ЯДРА

Ядро ОСРВ обеспечивает функционирование промежуточного абстрактного уровня ОС, который скрывает от прикладного ПО специфику технического устройства процессора (нескольких процессоров) и связанного с ним аппаратного обеспечения.

ОТЛИЧИЯ ОТ ОС ОБЩЕГО НАЗНАЧЕНИЯ

Однако ключевым отличием сервисов ядра ОСРВ является *детерминированный*, основанный на строгом контроле времени, характер их работы. В данном случае под детерминированностью понимается то, что для выполнения одного сервиса операционной системы требуется временной интервал заведомо известной продолжительности. Теоретически это время может быть вычислено по математическим формулам, которые должны быть строго алгебраическими и не должны включать никаких временных параметров случайного характера. Любая случайная величина, определяющая время выполнения задачи в ОСРВ, может вызвать нежелательную задержку в работе приложения, тогда следующая задача не уложится в свой квант времени, что послужит причиной для ошибки.

В этом смысле операционные системы общего назначения не являются детерминированными. Их сервисы могут допускать случайные задержки в своей работе, что может привести к замедлению ответной реакции приложения на действия пользователя в заведомо неизвестный момент времени. При проектировании обычных операционных систем разработчики не акцентируют своё внимание на математическом аппарате вычисления времени выполнения конкретной задачи и сервиса. Это не является критичным для подобного рода систем.

Выполнение задачи

В обычных ОСРВ задача может находиться в трёх возможных состояниях:

- задача выполняется;
- задача готова к выполнению;
- задача заблокирована.

Большую часть времени основная масса задач заблокирована. Только одна задача может выполняться на центральном процессоре в текущий момент времени. В примитивных ОСРВ список готовых к исполнению задач, как правило, очень короткий, он может состоять не более чем из двух-трёх наименований.

Основная функция администратора ОСРВ заключается в составлении такого планировщика задач.

Если в списке готовых к выполнению задач последних имеется не больше двух—трёх, то предполагается, что все задачи расположены в оптимальном порядке. Если же случаются такие ситуации, что число задач в списке превышает допустимый лимит, то задачи сортируются в порядке приоритета.

6. Графический интерфейс в Unix-системах. (НЕ НАПИСАНО, ВЗЯТО С КОЛКА)

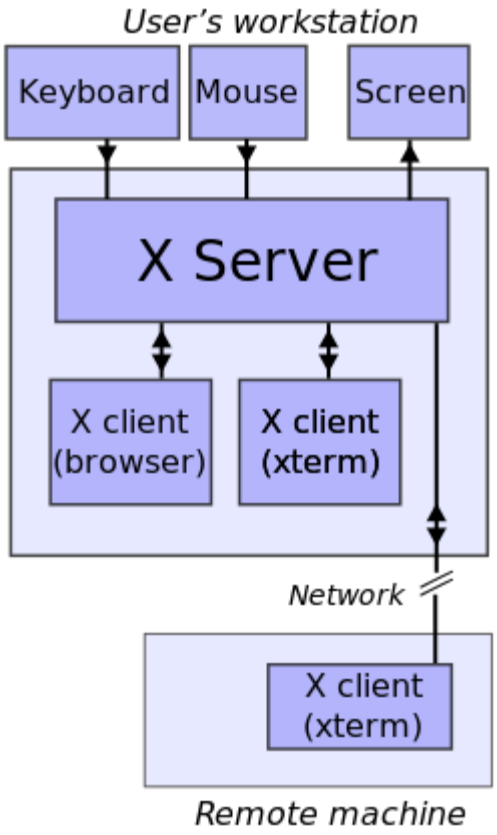
Система X Window System была разработана в Массачусетском технологическом институте (MIT) в 1984 году. По состоянию на февраль 2016 года версия протокола — ***X11*** — появилась в июне **2012 года**. Проект X возглавляет фонд X.Org Foundation. Референсная (или образцовая) реализация системы свободно доступна на условиях лицензии MIT и подобных ей лицензий. X Window System часто называют **X11** или просто **X** (в разговорной речи — *«иксы́»*).

X Window System — *оконная система*, обеспечивающая стандартные инструменты и протоколы для построения графического интерфейса пользователя.
X Window System обеспечивает базовые функции графической среды: отрисовку и перемещение окон на экране, взаимодействие с устройствами ввода, такими как, например, мышь и клавиатура. X Window System не определяет деталей интерфейса пользователя — этим занимаются менеджеры окон, которых разработано множество. По этой причине внешний вид программ в среде X Window System может очень сильно различаться в зависимости от возможностей и настроек конкретного оконного менеджера.

X Window System использует **клиент-серверную модель**: ***X-сервер*** обменивается сообщениями с различными ***клиентскими*** программами. Сервер принимает запросы на вывод графики (окон) и отправляет обратно пользовательский ввод (от клавиатуры, мыши или сенсорного экрана). X-сервер может быть:

- системной программой, управляющей выводом видео на персональном компьютере;
- приложением, отображающим графику в окно какой-то другой дисплейной системы;
- выделенным компонентом аппаратного обеспечения.

Пользовательский терминал в качестве «сервера» и удалённые приложения в качестве «клиентов» Локальный дисплей *предоставляет услуги отображения графики* программам и потому выступает в роли сервера. Удалённые программы *используют* эти услуги и потому играют роль клиентов.



В этом примере X-сервер принимает ввод с клавиатуры и мыши и производит вывод на экран. На пользовательской рабочей станции выполняются веб-браузер и эмулятор терминала. Программа обновления системы работает на удалённом сервере, но управляется с машины пользователя. Приложение работает так же, как если бы оно выполнялось локально.

Протокол, с помощью которого общаются сервер и клиент, является прозрачным для сети: клиент и сервер могут находиться как на одной и той же машине, так и на

разных. В частности, они могут работать на различных архитектурах под управлением разных операционных систем — результат будет одинаковым. Клиент и сервер могут даже безопасно взаимодействовать через Интернет посредством туннелирования соединения сквозь зашифрованный сетевой сеанс.

Xlib — библиотека функций клиента системы X Window, написанная на языке Си. Содержит функции для взаимодействия с т. н. X-сервером. Библиотека позволяет использовать более высокий уровень абстракции, без знания деталей работы основного протокола системы X Window.

Реализации X Window system

- Вплоть до 2004 года проект **XFree86** был наиболее распространённым вариантом X для свободных UNIX-подобных операционных систем. XFree86 возник как порт X на 386-совместимые персональные компьютеры. К концу 1990-х этот проект стал главным источником технических инноваций в X Window System и *де-факто* руководил разработкой X. Однако в 2004 году XFree86 поменял условия лицензии, и реализация X.Org Server (которая является форком XFree86, но со свободной лицензией) стала более распространённой.

- Референсная (эталонная) реализация от фонда X.Org Foundation, называемая **X.Org Server**, является канонической реализацией X Window System. Поскольку она распространяется на условиях весьма либеральной лицензии, появились несколько её разновидностей (как свободных, так и проприетарных).

X-терминал — это выделенное аппаратное обеспечение, на котором выполняется X-сервер и которое служит в качестве тонкого клиента. Эта архитектура завоевала популярность при построении недорогих терминальных парков, в которых множество пользователей одновременно используют один большой сервер приложений. Такое применение X Window System хорошо соответствует изначальным намерениям разработчиков из MIT. X-терминалы могут изучать сеть (в пределах локального широковещательного домена) с использованием протокола XDMCP, составляя при этом список узлов сети, с которых они могут запускать клиенты. На изначальном узле должен выполняться **дисплейный менеджер X** .

Графический дисплейный менеджер для X Window System (Display Manager) — это программа, которая регистрирует пользователей в операционной системе, предлагая им ввести их логин и пароль, которая также позволяет выбирать графическую среду (сеанс) и локаль (набор языковых и региональных настроек). Некоторые представители:

- XDM
- GDM
- KDM
- LightDM
- SDDM

X Window System намеренно не определяет, как должен выглядеть интерфейс пользователя приложения — кнопки, меню, заголовки окон и т. д. Эти вопросы решаются на уровне оконных менеджеров, инструментариев элементов интерфейса, сред рабочего стола и на уровне отдельных приложений.

Оконный менеджер управляет размещением и внешним видом окон приложений. Он может создавать интерфейс, подобный Microsoft Windows или Macintosh (например, так работают оконные менеджеры **Kwin в KDE** и **Metacity в GNOME**), или совершенно другой стиль (например, во фреймовых оконных менеджерах, таких как Ion).

Многие пользователи используют X вместе с полной **средой рабочего стола**, которая включает в себя оконный менеджер, различные приложения и единый стиль интерфейса. Наиболее популярные среды рабочего стола — **GNOME и KDE, XFCE, LXDE .**

Конкуренты X

*Компания Intel занята развитием нового протокола **Wayland**, который планируется как альтернатива X11, но при этом, в отличие от X11, не предоставляет средств удалённого доступа к приложениям на уровне протокола.*

***Mir** — это графический сервер для операционных систем на базе Linux, находящийся в разработке Canonical Ltd. Предполагалось, что он заменит X Window System в Ubuntu, однако 5 апреля 2017 года компания Canonical Ltd сообщила о прекращении работ над Mir.*

7. Основы работы в ОС Linux и пакет BusyBox.

BusyBox — набор UNIX-утилит командной строки, используемой в качестве основного интерфейса во встраиваемых операционных системах. Преимуществами этого приложения являются малый размер и низкие требования к аппаратуре. Оно представляет собой единый файл (это позволяет сэкономить дисковое пространство). Разработка BusyBox была начата в 1996 году Брюсом Перенсом.
ДЛЯ ЧЕГО НУЖЕН?

Проект BusyBox призван заменить одним исполняемым файлом традиционный для Unix-подобных систем «тяжеловесный» набор консольных команд и утилит проекта GNU, а также других OpenSource-программ. Во многих «прошивках» (firmware) встраиваемых систем BusyBox является основным средством командной строки, а также альтернативой веб-интерфейсу и другим методам администрирования.
Сейчас проект позиционируется как «швейцарский нож для встраиваемых Linux-систем» и в первую очередь ориентирован на использование в небольших дистрибутивах GNU/Linux. Распространяется под второй версией лицензии GNU GPL.
BusyBox представляет собой единственный исполняемый файл, при запуске которого загружается полноценный командный интерфейс. Как гласит официальная документация BusyBox, для формирования минимальной версии GNU/Linux к этому достаточно добавить лишь ядро ОС и каталоги /etc, /dev.
Утилита BusyBox оптимизирована с учетом ограниченных ресурсов среды выполнения. Модульная структура позволяет добавлять или удалять команды и опции на этапе компиляции, что позволяет «кастомизировать» встроенную систему по собственному вкусу. Разумеется, BusyBox предоставляет меньше функциональности, чем полновесные GNU-аналоги, реализуя лишь самые распространенные опции и поведение традиционного окружения Unix.
Утилита BusyBox ориентирована на Linux, однако большая часть кода является независимой от платформы, что, в принципе, позволяет «портировать» программу в другие UNIX-подобные ОС. С ядром Linux версии 2.6.x апплеты(тут это утилиты команд) BusyBox могут быть «собраны» для любой аппаратной архитектуры, которую поддерживает компилятор gcc.
Основу консольной среды BusyBox составляет командный интерпретатор (shell) ash (Almquist shell), в частности, используемый по умолчанию большинством BSD-систем. Присутствует и альтернативный вариант — интерпретатор hush (hyper utility shell). Вторым важным компонентом любой UNIX-подобной системы является так называемый пакет базовых утилит GNU (Core Utilities), включающий средства работы с файлами, текстом и программы, расширяющие возможности интерпретатора. В BusyBox реализованы аналоги многих базовых утилит: для работы с файлами и каталогами (cp, mkdir, mv, rm, rmdir, touch, mkfifo, mknod, sync, pwd и т. д.), текстовыми файлами (cat, cut, split, sort, tail, head, uniq, wc и т.д.), символьными и «жесткими» ссылками (ls, ln), управления правами (chgrp, chmod, chown), переменными окружения (env, printenv), просмотра свободного места на диске (du) и т.д. Работа с архивами и сжатие данных обеспечивается в BusyBox аналогами отдельных GNU-утилит: cpio, tar, gzip. Для управления задачами по расписанию используется реализация подсистемы cron; для управления процессами — ps, kill, nice; модулями ядра — lsmod, insmod, rmmod; для управления дисками и файловыми системами — fdisk, mkfs, mount и т.д. Среди возможностей BusyBox следует отметить разностороннюю поддержку сети, что существенно упрощает управление сетевым оборудованием, в основе системного ПО которого лежит ядро Linux.

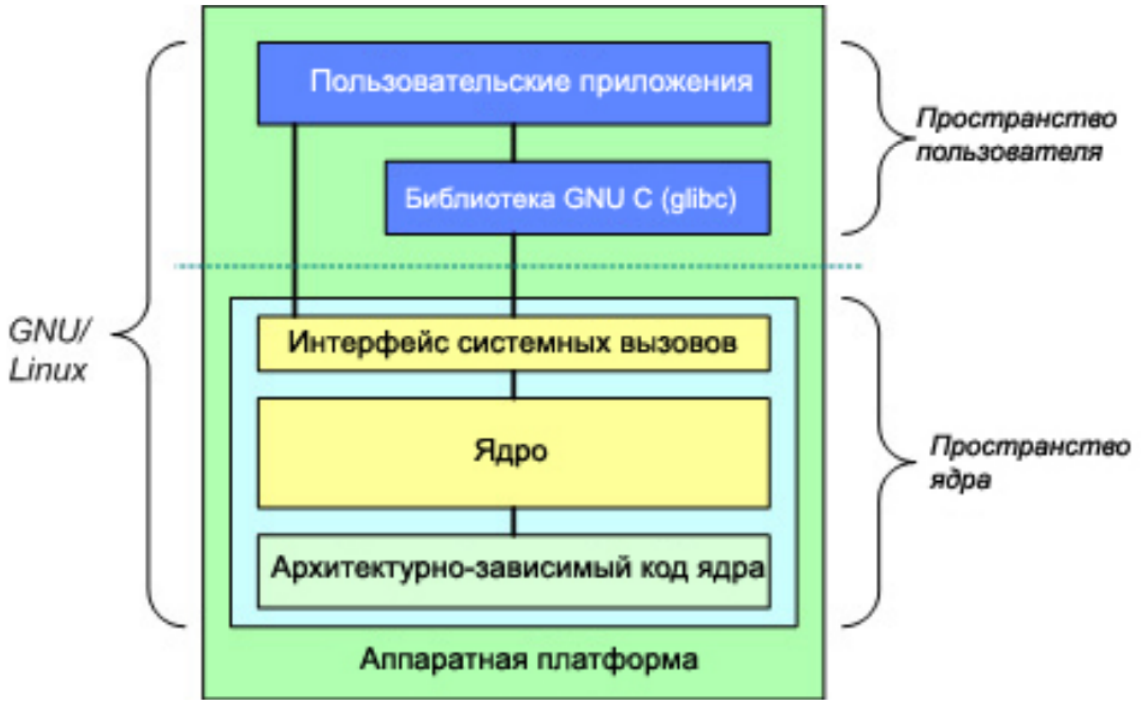
Следует также отметить, что конкурентов у рассматриваемой утилиты не так уж и много, и их функциональные возможности существенно меньше. Именно эти факторы и являются причиной популярности BusyBox как одного из ключевых компонентов системного ПО на основе ядра Linux для встраиваемых систем
BusyBox — набор UNIX-утилит командной строки, используемой в качестве основного интерфейса во встраиваемых операционных системах.
Преимущества:
▪ Малый размер.
▪ Низкие требования к аппаратуре.
Использование:

```
$ busybox ls -la
$ busybox ping 192.168.10.1
$ busybox ifconfig
```

8. Архитектура ядра Linux.

Ядро́ (kernel) — центральная часть операционной системы (ОС), обеспечивающая приложениям координированный доступ к ресурсам компьютера, таким как процессорное время, память, внешнее аппаратное обеспечение, внешнее устройство ввода и вывода информации.

Уровни ядра Linux:



Свойства ядра Linux

Ядро можно подразделить на множество различных подсистем
Все базовые сервисы собраны в ядре системы
Переносимость на различные платформы
Подсистемы ядра Linux:

(*кратко*)Блок управления процессами, Блок управления памятью, Драйверы устройств, Драйверы файловых систем, Блок управления сетью, Интерфейс системных вызовов
(*более подробно*)
Process Scheduler (SCHED) – планировщик процессов, отвечает за контроль над доступом процессов к CPU. Планировщик обеспечивает такое поведения ядра, при котором все процессы имеют справедливый доступ к центральному процессору.
Memory Manager (MM) – менеджер памяти, обеспечивает различным процессам безопасный доступ к основной памяти системы. Кроме того, MM обеспечивает работу виртуальной памяти, которая позволяет процессам использовать больше памяти, чем реально доступно в системе. Выделенная, но неиспользуемая память вытесняется на файловую систему, и при необходимости – возвращается из неё обратно в память (swapping).
Virtual File System(VFS) – виртуальная файловая система, создаёт абстрактный слой,

скрывая детали оборудования, предоставляя общий файловый интерфейс для всех устройств. Кроме того, VFS поддерживает несколько форматов файловых систем, которые совместимы с другими операционными системами.
Network Interface (NET) – сетевые интерфейсы, обеспечивает работу с различными сетевыми стандартами и сетевым оборудованием.
Inter-Process Communication (IPC) – межпроцессная подсистема, поддерживающая несколько механизмов для process-to-process связей в единой Linux-системе.

9. Модули ядра Linux.

ядро Linux *динамически изменяемое* – это означает, что вы можете загружать в ядро дополнительную функциональность, выгружать функции из ядра и даже добавлять новые модули, использующие другие модули ядра. Преимущество загружаемых модулей - возможность сократить расход памяти для ядра, загружая только необходимые модули (это может оказаться важным для встроенных систем)**.**

Таким образом, **модуль ядра** — это часть кода ядра, которая может быть динамически загружена и выгружена во время работы ядра.

Модуль ядра, загружаемый модуль ядра (loadable kernel module, LKM) — объект, содержащий код, который расширяет функциональность запущенного или т. н. базового ядра ОС.

Ядро Linux является монолитным. Это значит, что весь исполняемый код сосредоточен в одном файле. Такая архитектура имеет некоторые недостатки, например, невозможность установки новых драйверов без пересборки ядра. Но разработчики нашли решение и этой проблеме, добавив систему модулей.

Ядро Linux позволяет драйверам оборудования, файловых систем, и некоторым другим компонентам быть скомпилированными отдельно - как модули, а не как часть самого ядра. Таким образом, вы можете обновлять драйвера не пересобирая ядро, а также динамически расширять его функциональность. А еще это значит, что вы можете включить в ядре только самое необходимое, а все остальное подключать с помощью модулей.

Находятся все модули в папке /lib/modules/. Учитывая, что модули рассчитаны только для определенной версии ядра, то в этой папке создается отдельная подпапка, для каждой установленной в системе версии ядра.

Основные команды для управления модулями.

- **lsmod - посмотреть загруженные модули**
- **modinfo - информация о модуле**
- **insmod - загрузить модуль**
- **rmmod - удалить модуль**

Работа с модулями ядра Linux выполняется, в основном, с помощью этих команд, но могут использоваться и другие.

DKMS - фреймворк, который используется для генерации тех модулей ядра Linux, которые в общем случае не включены в дерево исходного кода. **Причем модуль, установленный таким образом один раз, будет пересобираться для каждой новой версии ядра автоматически.**

Структура каталогов модулей ядра

В ядрах серии 2.6 в kernel/ обычно имеются следующие подкаталоги:

- arch — модули ядра, зависящие от архитектуры;
- crypto — криптографические модули ядра;
- drivers — драйверы для различных устройств;
- fs — модули поддержки файловых систем;
- kernel — служебный модуль;
- lib — библиотеки, используемые ядром;
- net — поддержка сетевой инфраструктуры;
- sound — поддержка звуковой инфраструктуры.

Анатомия модуля ядра

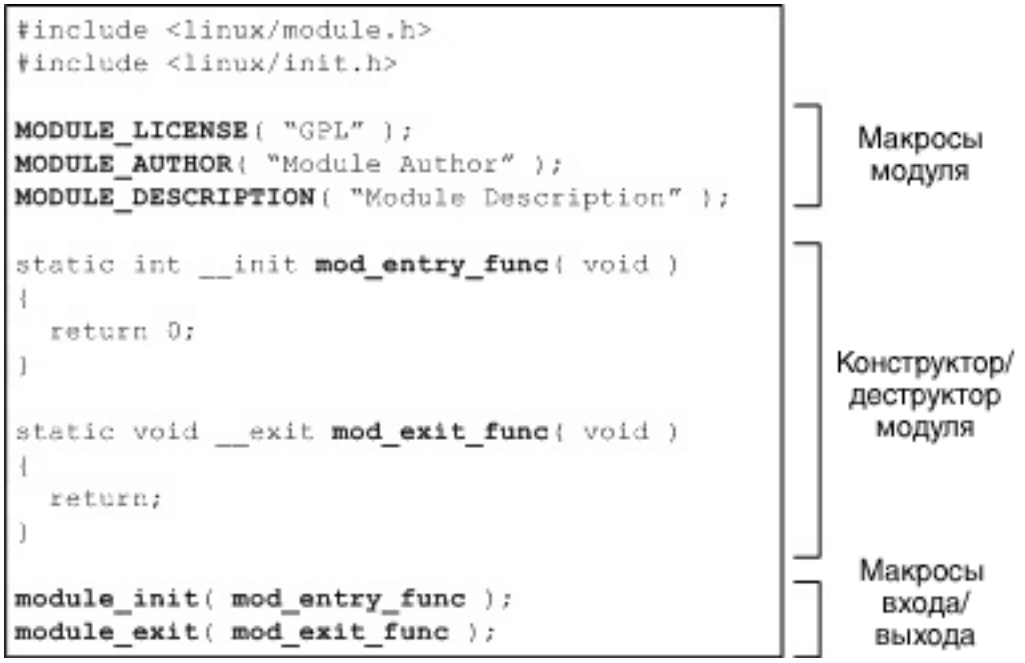
Загружаемые модули ядра имеют ряд фундаментальных отличий от элементов, интегрированных непосредственно в ядро, а также от обычных программ: обычная программа содержит main, а *загружаемые модули, содержат функции входа и выхода*.

Функция входа - когда модуль загружается в ядро

Функция выхода – соответственно при выгрузке из ядра

Поскольку функции входа и выхода являются пользовательскими, для указания назначения этих функций используются макросы module_init и module_exit . Загружаемый модуль содержит также набор обязательных и дополнительных макросов. Они определяют тип лицензии, автора и описание модуля, а также другие параметры. Пример очень простого загружаемого модуля приведен на рисунке 1.

Рисунок 1. Код простого загружаемого модуля.



Версия 2.6 ядра Linux предоставляет новый, более простой метод создания загружаемых модулей. После того как модуль создан, можно использовать обычные пользовательские инструменты для управления модулями (несмотря на изменения внутреннего устройства): insmod (устанавливает модуль), rmmod (удаляет модуль), modprobe (контейнер для insmod и rmmod), depmod (для создания зависимостей между модулями) и modinfo (для поиска значений в модулях макроса).

10.Типы архитектур ядер операционных систем.

Выбор архитектуры ядра играет очень важную, можно сказать, первичную роль при разработке операционной системы, так как именно архитектура ядра определяет, как компоненты операционной системы будут взаимодействовать между собой.

Обычно выделяют следующие *основные* типы архитектуры ядра:

- **монолитное**, или макроядро (может различаться на «классическое» монолитное и модульное ядро);
- **микроядро**.

Выделяют *производные* архитектуры ядра:

- **Модульное ядро**
- **Экзоядро**
- **Наноядро**
- **Гибридное ядро**

Монолитное ядро:

Монолитное ядро предоставляет богатый набор абстракций оборудования. Все части монолитного ядра работают в *одном адресном* пространстве. Большинство современных ядер позволяют во время работы подгружать модули, выполняющие части функции ядра.

Достоинства: Скорость работы, упрощённая разработка модулей, богатство предоставляемых возможностей и функций, поддержка большого количества разнообразного оборудования.

Недостатки: Поскольку всё ядро работает в одном адресном пространстве, сбой в одном из компонентов может нарушить работоспособность всей системы.

В этом случае компоненты ОС являются не самостоятельными модулями, а составными частями одной большой программы. Такая структура операционной системы называется монолитным ядром (monolithic kernel). Монолитное ядро представляет собой набор процедур, каждая из которых может вызвать каждую. Все процедуры работают в привилегированном режиме.

Таким образом, монолитное ядро — это такая схема операционной системы, при которой все её компоненты являются составными частями одной программы, используют общие структуры данных и взаимодействуют друг с другом путём непосредственного вызова процедур. Для монолитной операционной системы ядро совпадает со всей системой.

Примеры: IBM OS/360, RSX-11, VAXVMS, MS-DOS, Windows (все модификации), OS/2, Linux, все клоны BSD (FreeBSD, NetBSD, OpenBSD), Sun Solaris.

Микроядро:

Микроядро предоставляет только элементарные функции управления процессами и минимальный набор абстракций для работы с оборудованием. Большая часть работы осуществляется с помощью специальных пользовательских процессов, называемых сервисами.

Достоинства: Устойчивость к сбоям оборудования, ошибкам в компонентах системы.

Недостатки: Передача данных между процессами требует накладных расходов.

Современная тенденция в разработке операционных систем состоит в перенесении значительной части системного кода на уровень пользователя и одновременной минимизации ядра. Речь идет о подходе к построению ядра, называемом микроядерной архитектурой (microkernel architecture) операционной системы, когда большинство ее составляющих являются самостоятельными программами. В этом случае взаимодействие между ними обеспечивает специальный модуль ядра, называемый микроядром. Микроядро работает в привилегированном режиме и обеспечивает взаимодействие между программами, планирование использования процессора, первичную обработку прерываний, операции ввода-вывода и базовое управление памятью. Остальные компоненты системы взаимодействуют друг с другом путем передачи сообщений через микроядро.

Примеры: Symbian OS; Mach, используемый в GNU/Hurd и Mac OS X; Windows CE; QNX; AIX; Minix ; ChorusOS ; AmigaOS; MorphOS.

Модульное ядро

Модульное ядро — современная, усовершенствованная модификация архитектуры монолитных ядер операционных систем компьютеров. Модульные ядра не требуют полной перекомпиляции ядра при изменении состава аппаратного обеспечения компьютера. Вместо этого модульные ядра предоставляют тот или иной механизм ····подгрузки модулей ядра, поддерживающих то или иное аппаратное обеспечение (например, драйверов). При этом подгрузка модулей может быть как динамической (выполняемой «на лету», без перезагрузки ОС, в работающей системе), так и

статической (выполняемой при перезагрузке ОС после переконфигурирования системы на загрузку тех или иных модулей).

Все модули ядра работают в адресном пространстве ядра и могут пользоваться всеми функциями, предоставляемыми ядром. Поэтому модульные ядра продолжают оставаться монолитными. Модульность ядра осуществляется на уровне бинарного образа, т.к. динамически подгружаемые модули загружаются в адресное пространство ядра и в дальнейшем работают как интегральная часть ядра. Модули позволяют легко расширить возможности ядра по мере необходимости.

Примером может служить VFS — «виртуальная файловая система», совместно используемая многими модулями файловых систем в ядре Linux.

Экзоядро

Экзоядро — ядро ОС компьютеров, предоставляющее лишь функции для взаимодействия между процессами и безопасного выделения и освобождения ресурсов. Предполагается, что API для прикладных программ будут предоставляться внешними по отношению к ядру библиотеками (откуда и название архитектуры). Возможность доступа к устройствам на уровне контроллеров позволит эффективней решать некоторые задачи, которые плохо вписываются в рамки универсальной ОС, например реализация СУБД будет иметь доступ к диску на уровне секторов диска, а не файлов и кластеров, что положительно скажется на быстродействии.

Наноядро

Наноядро — архитектура ядра операционной системы компьютеров, в рамках которой крайне упрощённое и минималистичное ядро выполняет лишь одну задачу — обработку аппаратных прерываний, генерируемых устройствами компьютера. После обработки прерываний от аппаратуры наноядро, в свою очередь, посылает информацию о результатах обработки (например, полученные с клавиатуры символы) вышележащему программному обеспечению при помощи того же механизма прерываний. Примером является KeyKOS — самая первая ОС на наноядре. Первая версия вышла ещё в 1983-ем году.

Гибридное ядро

Гибридные ядра — это модифицированные микроядра, позволяющие для ускорения работы запускать «несущественные» части в пространстве ядра. Смешанное ядро, в принципе, должно объединять преимущества монолитного ядра и микроядра: казалось бы, микроядро и монолитное ядро — крайности, а смешанное — золотая середина. В них возможно добавлять драйвера устройств двумя способами: и внутрь

ядра, и в пользовательское пространство. Но на практике концепция смешанного ядра часто подчёркивает не только достоинства, но и недостатки обоих типов ядер.

Примеры: Windows NT, DragonFlyBSD.

11. Установка программного обеспечения в Linux.

Пакетные системы в Linux

Дистрибутив	**Низкоуровневые инструменты**	**Высокоуровневые инструменты**
Debian	dpkg	apt-get / aptitude
RHEL (CentOS)	rpm	yum
openSUSE	rpm	zypper

Пакетные менеджеры в ОС Linux

Operating System	Format	Tool(s)
Debian	`.deb`	`apt`, `apt-cache`, `apt-get`, `dpkg`
Ubuntu	`.deb`	`apt`, `apt-cache`, `apt-get`, `dpkg`
CentOS	`.rpm`	`yum`
Fedora	`.rpm`	`dnf`
FreeBSD	Ports, `.txz`	`make`, `pkg`

Система управления пакетами APT

APT - Advanced Packaging Tool - представляет собой обертку над утилитой dpkg, с помоўью которой осуществляется поиск пакетов, сверка контрольных сумм, выкачивание из репозитория, разрешение зависимостей и другие действия.

Основные понятия apt

- **Пакет** - программа, библиотека, исходники или любые другие файлы, а также их метаданные, упакованные в особым образом сформированный архив.
- **Репозиторий** - место хранения deb-пакетов.
- Кэш доступных пакетов
- Индексный файл
- Apt keyring

Структура пакета deb

- data.tar.gz
- control.tar.gz
- debian-binary

$ ar -xv пакет.deb && tar -xzf data.tar.gz

Конфигурационные файлы apt

- /etc/apt/sources.list
- /etc/apt/apt.conf
- /etc/apt/preference

Основные команды apt

- apt-cache search [mask]
- apt-cache show [package]
- apt-get install [package]
- apt-get remove [package] или apt-get install --remove [package]
- apt-get purge [package]
- apt-get update
- apt-get upgrade
- apt-get autoremove

apt-get add-apt-repository “deb URL” или

apt-get add-apt-repository ppa:[name ppa]

Установщики пакетов

- Консольные утилиты
    - apt-get
    - aptitude
    - tasksel
    - deselect
    - dpkg-deb
    - dpkg-split
    - dpkg
- Графические менеджеры
    - Update Manager

- Ubuntu Software Center или Ubuntu Software (с Ubuntu 16.04)
- Gdebi
- Synaptic
- Другие менеджеры

Сторонние источники пакетов: правила

1. Добавить ссылку на репозиторий в файл /etc/apt/sources.list в следующем формате:

deb uri дистрибутив
[компонент1] [компонент2] [...]

2.Добавить в apt keyring публичный ключ репозитория:

$ sudo apt-key adv --keyserver сервер-сертификации --recv-keys ID-ключа

3. Обновить кэш доступных пакетов:

$ sudo apt-get update

4. Установить пакет:

apt-get install пакет

Утилита dpkg

Команда	**Пояснение**
dpkg -i package.deb	Установка пакета
dpkg -l	Просмотр всех установленных пакетов
dpkg -l apache2	Просмотр установлен или нет пакет apache2
dpkg -r package.deb	Удаление пакета
dpkg -p package.deb	Удаление пакета с конфигурационными файлами
dpkg -c package.deb	Просмотр содержимого пакета

Преимущества менеджеров пакетов

- одни и те же библиотеки используются для нескольких программ, что экономит место;
- понятная файловая структура, да и, вообще, unix-way;

- конкретный разработчик обеспечивает работоспособность отдельного пакета;
- обеспечивается целостность системы;
- программа устанавливается один раз и для всех пользователей.

Недостатки менеджеров пакетов

- старые версии пакетов в репозиториях, а если нужно свежее и новее, то приходится использовать PPA или другие средства;
- зависимость от конкретных версий библиотек, поставляемых совместно с системой;
- сложность установки программ разных версий;
- приходится скачивать большое количество зависимостей, которые зачастую не столь необходимых;
- необходимо устанавливать программы из-под root-пользователя;

Что такое snap?

- snap - новый формат пакетов, разработанный в Canonical.
- snap-пакеты же содержат и саму программу, и все её зависимости.

Преимущества snap

- для установки пакета не нужны права root-пользователя;
- нет проблем с версиями библиотек;
- гарантируется работа не только отдельного приложения, но и его работу с библиотеками;
- упрощается жизнь производителям проприетарного программного обеспечения;
- полностью отсутствуют зависимости от библиотек системы.

Недостатки snap

- увеличение места на носителе информации, каждый пакет таскает с собой свои библиотеки;
- усложнение файловой структуры Ubuntu (для каждого пакета — своя папка);
- отход от unix-way;
- нарушается целостность системы.

Команды snap

$ sudo snap install пакет

$ sudo snap refresh пакет

$ sudo snap remove пакет

Ubuntu-make

AppImage

- Скачать пакет Appimage

- Назначить права на выполнение
- Запустить на выполнение

Flatpak

- sudo apt install flatpak
- sudo apt install gnome-software-plugin-flatpak
- flatpak remote-add --if-not-exists flathub https://flathub.org/repo/flathub.flatpakrepo
- flatpak install <remote_repository> <application_id>
- flatpak install flathub org.libreoffice.LibreOffice

12.Типы файлов и управление разрешениями на доступ к файлам.

Виртуальная файловая система VFS поддерживает следующие типы файлов:

▪ Обычные файлы -
▪ Каталоги
▪ Специальные файлы
▪ Именованные конвейеры
▪ Символьные связи

- *Регулярный файл* - это обычный файл, который содержит разнообразные данные или команды.
- *Каталоги* - это файлы, которые содержат указатели на другие файлы.
- *Файлы символьных и блочных устройств* - системные файлы, предназначенные для организации обмена данными между такими устройствами.
- *Локальные сокеты* - это файлы, которые позволяют организовать обмен данными между локальными процессами в двух направлениях.
- *Именованные каналы или FIFO* - это файлы, которые позволяют организовать обмен данными между локальными процессами в одном направлении.
- *Символическая ссылка* - это файл, который содержит абсолютный или относительный путь к другому файлу или каталогу

Также существует разделение на типы файлов в архивах:

Символ	Пояснение	Примечание
-	Обычный файл	Сюда входят текстовые файлы, файлы данных, исполняемые файлы и т. д.
d	Каталог	
c	Символьное устройство	Особый файл, используемый для взаимодействия с драйвером символьного устройства. Традиционно эти файлы сосредоточены в каталоге /dev; в архивах они обычно отсутствуют
b	Блочное устройство	Особый файл, применяемый для взаимодействия с драйвером блочного устройства. По традиции эти файлы сосредоточены в каталоге /dev; в архивах они обычно отсутствуют
l	Символическая ссылка	Имя файла, указывающее на другое имя файла. Файл, на который оно указывает, может располагаться в другой файловой системе либо вообще не существовать

- s - файл локального сокета (Local domain socket);
- p - именованный канал (Named pipe);

Модель разрешений на доступ к файлам:

Концепции модели разрешений на доступ к файлам:

- классы пользователей для доступа к файлу;

- разрешения на доступ к файлу.

Пользователи:

- владелец файла;

- группа файла;

- другие пользователи системы.

Специальные разрешения:

- setuid или SUID - режим set user ID
- setgid или SGID - режим set group ID
- sticky bit - sticky (липкий) режим

Изменение разрешений на доступ:

chmod [OPTIONS] MODE[,MODE]... FILE...

chmod [OPTIONS] OCTAL-MODE FILE...

chmod [OPTIONS] --reference=RFILE FILE...

где

- OPTIONS - опции команды;

- MODE - режимы доступа в текстовых обозначениях;

- OCTAL-MODE - режимы доступа в числовых обозначениях;

- FILE - имя файла.

chmod [класс] [изменение] [права] [файлы]

u — владелец g — группа o — все остальные a — все	u — владелец g — группа o — все остальные a — все	u — владелец g — группа o — все остальные a — все

Примеры:
chmod gw,a+r file
chmod ug=rw,o= file

или

```
chmod 644 report.txt
```

Установка маски разрешений:
umask [-S] [mask-definition]

где

- опция -S указывает на то, что маска разрешений задана или должна быть отображена в символьном формате;

- аргумент mask-definition задает маску разрешений на доступ к файлу или каталогу в числовом или символьном формате.

Пример: `umask 200`

Права на каталоги должны быть нечетными, либо отсутствовать:

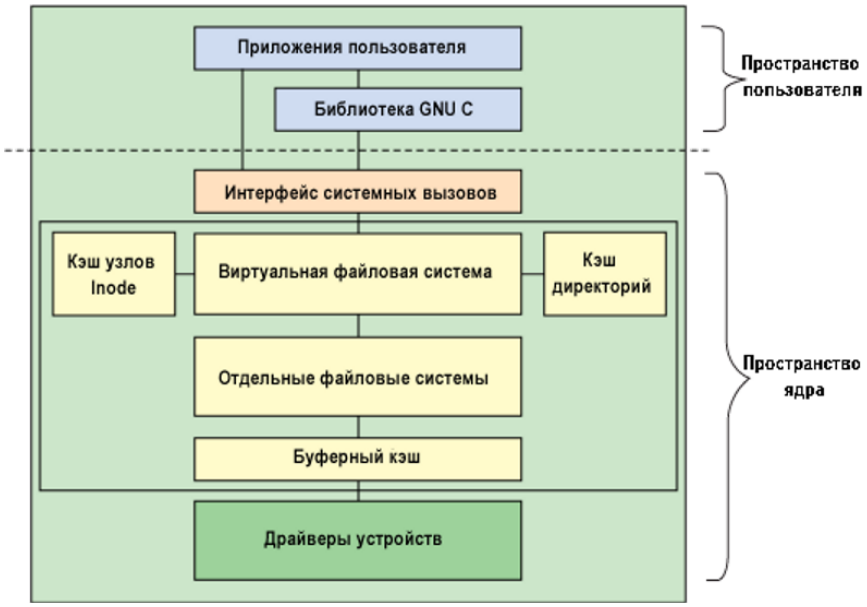
▪0 — прав нет;
▪1 (x) — можно входить в каталог и обращаться к файлам в нем;
▪3 (wx) — можно входить в каталог, разрешены операции с файлами,
▪однако получить список файлов нельзя;
▪5 (rx) — можно входить в каталог, получать подробную информацию о файлах обращаться к файлам, нельзя переименовывать и удалять файлы;
▪7 (rwx) — полные права;

13.Концепция файловой системы Linux.

Файловая система – это подсистема операционной системы, которая управляет доступом к файлам, расположенным на устройствах внешней памяти.

Процесс связывания файловой системы с устройством в Linux называется *монтированием* (*mounting*). Для подключения файловой системы к существующей иерархии файловых систем (корню) используется команда mount. При монтировании указывается файловая система, ее тип и точка монтирования.

Архитектура высокого уровня
Хотя большая часть кода файловой системы реализована в ядре (за исключением файловых систем пространства пользователя, о которых я расскажу ниже), архитектура, показанная на рисунке 1, характеризует отношения между основными компонентами файловой системы, как в пространстве пользователя, так и в ядре.
Рисунок 1. Архитектурное представление компонентов файловой системы Linux



В пространстве пользователя размещаются приложения (в этом примере - пользователь файловой системы) и библиотека GNU C (glibc), которые предоставляют интерфейс для вызова файловой системы (открытие, чтение, запись, закрытие). Интерфейс системных вызовов действует как коммутатор, направляющий системные вызовы из пространства пользователя в соответствующую точку пространства ядра.

VFS является основным интерфейсом к файловым системам нижнего уровня. Этот компонент экспортирует набор интерфейсов и после этого абстрагирует их в отдельные файловые системы, образ поведения которых может быть весьма различным. Для объектов файловой системы (узлов inodes и записей dentries) существуют два кэша, о которых я скоро расскажу. Каждый из них предоставляет пул недавно использованных объектов файловой системы.

Реализация каждой файловой системы, например, ext2, JFS и так далее, экспортирует общий массив интерфейсов, который используется (и ожидается) VFS. Буферный кэш буферизирует запросы между файловыми системами и блочными устройствами, которыми они могут управлять. Например, через буферный кэш проходят запросы на чтение и запись к драйверам устройств. Это позволяет кэшировать запросы для более быстрого доступа (вместо обращения к физическому устройству). Буферный кэш управляется набором списков последних использованных элементов (least recently used, LRU). Обратите внимание, что командой sync можно сбросить буферный кэш на носитель (принудительно отправить все незаписанные данные на драйверы устройств и, в дальнейшем, на устройства хранения).

Уровень виртуальной файловой системы

VFS действует как корневой уровень интерфейса файловой системы. VFS следит за всеми поддерживаемыми и всеми смонтированными на данный момент файловыми системами.

Файловые системы в Linux можно динамически добавлять и удалять с помощью нескольких функций регистрации.
Системным блоком (superblock) называется структура, представляющая файловую систему. В нее входит информация, необходимая для управления файловой системой во время работы. К такой информации относится название файловой системы (например, ext2), размер файловой системы и ее состояние, ссылку на блочное устройство и информацию метаданных (например, списки свободных блоков и т.п.).
Узел inode представляет объект файловой системы с уникальным идентификатором. Каждая файловая система предоставляет методы преобразования имен файлов в уникальные идентификаторы inode и затем – в ссылки на inode. Каждая из этих структур ссылается на отдельные операции, которые могут выполняться с inode. Например, inode_operations определяет операции, которые работают напрямую с inode, а file_operations относится к методам, которые работают с файлами и директориями (стандартные системные вызовы).
Иерархическая природа файловой системы управляется другим объектом в VFS, называемым ***dentry***. Объект dentry — это определенный компонент пути.Например, при открытии файла, составленного из /home/user/name, создается четыре dentry-объекта: один для корня /, один для записи home корневого каталога, один для записи name каталога user и, наконец, один для записи name в каталоге user. Таким образом, записи dentry четко отображаются на иерархические файловые системы, используемые сегодня.
Для каждого открытого в Linux-системе файла существует объект **file.** Этот объект содержит информацию, относящуюся к открытому экземпляру для данного пользователя.
Последние использованные узлы inodes и записи dentries хранятся в кэше inode и кэше директорий соответственно. Обратите внимание, что каждому inode в кэше inode соответствует запись dentry в кэше директорий.
Буферный кэш
За исключением отдельных реализаций файловых систем (которые можно найти в ./linux/fs), в нижней части уровня файловой системы располагается буферный кэш. Здесь хранятся запросы на чтение и запись от отдельных файловых систем и физических устройств (посредством драйверов устройств). Из соображений производительности в Linux предусмотрен кэш запросов, позволяющий не обращаться по каждому запросу к физическому устройству. Вместо этого в нем кэшируются последние использованные буферы (страницы), которые могут быть быстро предоставлены отдельным файловым системам.

14.Стандартные консольные программы и команды в ОС Linux. Оператор “|”, конвейер, перенаправление ввода-вывода

1. LS

Утилита для просмотра содержимого каталогов.

2. CAT

Печатает содержимое файла, переданного в параметре, в стандартный вывод.

3. CD

Позволяет перейти из текущего каталога в указанный.

4. PWD

Печатает на экран текущий каталог.

5. MKDIR

Создание новых каталогов.

6. FILE

Показывает тип файла. В Linux файлы не обязаны всегда иметь расширения для того, чтобы с ними работать. Поэтому пользователю иногда трудно определить, что за файл перед ним. Эта маленькая утилита решает проблему.

7. CP

Копирование файлов и каталогов. Она не копирует каталоги по умолчанию рекурсивно (то есть все поддиректории и все файлы в поддиректориях), поэтому не забудьте добавить опцию -r (Recursive) или -a (Archive). Последняя включает режим сохранения атрибутов, владельца и временного штампа в дополнение к рекурсивному копированию.

8. MV

Перемещение или переименование файлов и каталогов. Примечательно, что в Linux это одна и та же операция. Переименование - это перемещение файла в ту же папку с другим именем.

9. RM

Удаляет файлы и папки. Очень полезная команда Linux: с её помощью вы можете убрать весь беспорядок. Если нужно рекурсивное удаление, используйте опцию -r. Однако будьте осторожны: конечно, для того чтобы повредить систему вам нужно будет серьёзно постараться, однако можно удалить собственные важные файлы. Rm удаляет файлы не в корзину, из которой потом всё можно будет восстановить, а полностью стирает. Действия оператора rm необратимы. Поверьте, ваши оправдания в духе "rm съела мою курсовую" никому не будут интересны.

10. LN

Создает жёсткие или символические ссылки на файлы. Символические или программные ссылки - это что-то похожее на ярлыки в Windows. Они предоставляют удобный способ доступа к определённому файлу. Символические ссылки указывают на файл, но не имеют никаких метаданных. Жёсткие ссылки, в отличие от символических, указывают на физический адрес области диска, где хранятся данные файла.

11. CHMOD

Изменяет права доступа к файлу. Это чтение, запись и выполнение. Каждый пользователь может изменять права для своих файлов.

12. CHOWN

Изменяет владельца файла. Только суперпользователь может изменять владельцев. Для рекурсивного изменения используйте опцию -R.

13. FIND

Поиск в файловой системе, файлах и папках. Это очень гибкая и мощная команда Linux не только из-за своих способностей ищейки, но и благодаря возможности выполнять произвольные команды для найденных файлов.

14. LOCATE

В отличие от find, команда locate ведёт поиск в базе данных updatedb для шаблонов имён файлов. Эта база данных содержит снимок файловой системы, что позволяет искать очень быстро. Но этот поиск ненадёжен, потому что вы не можете быть уверены, что ничего не изменилось с момента последнего снимка.

15. DU

Показывает размер файла или каталога. Самые полезные опций: -h (Human), которая преобразует размеры файлов в легко читаемый формат, -s (Summarize), которая выводит минимум данных, и -d (Depth), устанавливающая глубину рекурсии по каталогам.

16. DF

Анализатор дискового пространства. По умолчанию вывод достаточно подробный: перечислены все файловые системы, их размер, количество использованного и свободного пространства. Для удобства есть опция -h, делающая размеры легко читаемыми.

17.DD

Как сказано в официальном руководстве, это команда терминала для копирования и преобразования файлов. Не очень понятное описание, но это всё, что делает dd. Вы передаёте ей файл-источник, пункт назначения и пару дополнительных опций. Затем она делает копию одного файла в другой. Вы можете задать точный размер данных, которые нужно записать или скопировать. Работает утилита со всеми устройствами. Например, если вы хотите перезаписать жёсткий диск нулями из /dev/zero, можете сделать это.

Также она часто используется для создания LiveUSB или гибридных ISO образов.

18 MOUNT / UMOUNT

Это команды консоли Linux для подключения и отключения файловых систем Linux. Можно подключать всё: от USB накопителей, до ISO образов. И только у суперпользователя есть права для этого.

LINUX КОМАНДЫ КОНСОЛИ ДЛЯ РАБОТЫ С ТЕКСТОМ

19. MORE / LESS

Это две простенькие команды терминала для просмотра длинных текстов, которые не вмещаются на одном экране. Представьте себе очень длинный вывод команды. Или вы вызвали cat для просмотра файла, и вашему эмулятору терминала потребовалось несколько секунд, чтобы прокрутить весь текст. Если ваш терминал не поддерживает прокрутки, вы можете сделать это с помощью less. Less новее, чем more и поддерживает больше опций, поэтому использовать more нет причин.

20. HEAD / TAIL

Ещё одна пара, но здесь у каждой команды своя область применения. Head выводит несколько первых строк из файла (голова), a tail выдает несколько последних строк (хвост). По умолчанию каждая утилита выводит десять строк. Но это можно изменить с помощью опции -n. Ещё один полезный параметр -f, это сокращение от follow (следовать). Утилита постоянно выводит изменения в файле на экран. Например, если вы хотите следить за лог файлом, вместо того, чтобы постоянно открывать и закрывать его, используйте команду tail -nf.

21. GREP

Grep, как и другие инструменты Linux, делает одно действие, но делает его хорошо: она ищет текст по шаблону. По умолчанию она принимает стандартный ввод, но вы можете искать в файлах. Шаблон может быть

строкой или регулярным выражением. Она может вывести как совпадающие, так и не совпадающие строки и их контекст. Каждый раз, когда вы выполняете команду, которая выдает очень много информации, не нужно анализировать всё вручную - пусть grep делает свою магию.

22. SORT

Сортировка строк текста по различным критериям. Наиболее полезные опции: -n (Numeric), по числовому значению, и -r (Reverse), которая переворачивает вывод. Это может быть полезно для сортировки вывода du. Например, если хотите отсортировать файлы по размеру, просто соедините эти команды.

23. WC

Утилита командной строки Linux для подсчёта количества слов, строк, байт и символов.

24. DIFF

Показывает различия между двумя файлами в построчном сравнении. Причём выводятся только строки, в которых обнаружены отличия. Измененные строки отмечаются символом "c", удалнные - "d", а новые - "a".

Кстати, я подготовил ещё одну подробную статью, в которой описан именно просмотр содержимого текстового файла в Linux с помощью терминала.

КОМАНДЫ LINUX ДЛЯ УПРАВЛЕНИЯ ПРОЦЕССАМИ

25. KILL / XKILL / PKILL / KILLALL

Служат для завершения процессов. Но они принимают различные параметры для идентификации процессов. Kill нужен PID процесса, xkill - достаточно кликнуть по окну, чтобы закрыть его, killall и pkill принимают имя процесса. Используйте ту, которая удобна в определенной ситуации.

26. PS / PGREP

Как уже говорилось, чтобы уничтожить процесс, нужен его идентификатор. Один из способов получить его, это утилита ps, которая печатает информацию о запущенных процессах. По умолчанию вывод очень длинный, поэтому используйте опцию -e, чтобы увидеть информацию об определённом процессе. Это только снимок состояния на момент вызова, и информация не будет обновляться. Команда ps с ключом aux выводит полную информацию о процессах. Pgrep работает следующим образом: вы задаете имя процесса, а утилита показывает его идентификатор.

27. TOP / HTOP

Обе команды похожи, обе отображают процессы и могут быть использованы как консольные системные мониторы. Я рекомендую установить htop, если в вашем дистрибутиве он не поставляется по умолчанию, так как это улучшенная версия top. Вы сможете не только просматривать, но и контролировать процессы через его интерактивный интерфейс.

28. TIME

Время выполнения процесса. Это секундомер для выполнения программы. Полезно, если вам интересно, насколько сильно ваша реализация алгоритма отстает от стандартной. Но, несмотря на такое название, она не сообщит вам текущее время, используйте для этого команду date.

КОМАНДЫ LINUX ОКРУЖЕНИЯ ПОЛЬЗОВАТЕЛЯ

29. SU / SUDO

Su и sudo - это два способа выполнить одну и ту же задачу: запустить программу от имени другого пользователя. В зависимости от вашего дистрибутива вы, наверное, используете одну или другую. Но работают обе. Разница в том, что su переключает вас на другого пользователя, а sudo только выполняет команду от его имени. Поэтому использование sudo будет наиболее безопасным вариантом работы.

30. DATE

В отличие от time, делает именно то, чего вы от неё и ожидаете: выводит дату и время в стандартный вывод. Его можно форматировать в зависимости от ваших потребностей: вывести год, месяц, день, установить 12-ти или 24-ти часовой формат, получить наносекунды или номер недели. Например, date +"%j %V", выведет день в году и номер недели в формате ISO.

31. ALIAS

Команда создаёт синонимы для других команд Linux. То есть вы можете делать новые команды или группы команд, а также переименовывать существующие. Это очень удобно для сокращения длинных команд, которые вы часто используете, или создания более понятных имен для команд, которые вы используете нечасто и не можете запомнить.

32. UNAME

Выводит некую основную информацию о системе. Без параметров она не покажет ничего полезного, кроме строчки Linux, но, если задать параметр -a (All), можно получить информацию о ядре, имени хоста и узнать архитектуру процессора.

33. UPTIME

Сообщает вам время работы системы. Не очень существенная информация, но может быть полезна для случайных вычислений или просто ради интереса, чтобы узнать, как давно был перезагружен сервер.

34. SLEEP

Вам, наверное, интересно как же её можно использовать. Даже не учитывая Bash-скриптинг, у неё есть свои преимущества. Например, если вы хотите выключить компьютер через определенный промежуток времени или использовать в качестве импровизированной тревоги.

КОМАНДЫ LINUX ДЛЯ УПРАВЛЕНИЯ ПОЛЬЗОВАТЕЛЯМИ

35. USERADD / USERDEL / USERMOD

Эти команды консоли Linux позволяют вам добавлять, удалять и изменять учетные записи пользователей. Скорее всего, вы не будете использовать их очень часто. Особенно если это домашний компьютер, и вы являетесь единственным пользователем. Управлять пользователями можно и с помощью графического интерфейса, но лучше знать об этих командах на всякий случай.

36. PASSWD

Эта команда позволяет изменить пароль учетной записи пользователя. Как суперпользователь вы можете сбросить пароли всех пользователей, даже несмотря на то, что не можете их увидеть. Хорошая практика безопасности - менять пароль почаще.

LINUX КОМАНДЫ ДЛЯ ПРОСМОТРА ДОКУМЕНТАЦИИ

37. MAN / WHATIS

Команда man открывает руководство по определённой команде. Для всех основных команд Linux есть man страницы. Whatis показывает, какие разделы руководств есть для данной команды.

38. WHEREIS

Показывает полный путь к исполняемому файлу программы. Также может показать путь к исходникам, если они есть в системе.

КОМАНДЫ LINUX ДЛЯ УПРАВЛЕНИЯ СЕТЬЮ

39. IP

Если список команд Linux для управления сетью вам кажется слишком коротким, скорее всего вы не знакомы с утилитой ip. В пакете net-tools

содержится множество других утилит: ipconfig, netstat и прочие устаревшие, вроде iproute2. Всё это заменяет одна утилита - ip. Вы можете рассматривать её как швейцарский армейский нож для работы с сетью или как непонятную массу, но в любом случае за ней будущее. Просто смиритесь с этим.

40. PING

Ping - это ICMP ECHO_REQUEST дейтаграммы, но на самом деле это неважно. Важно то, что утилита ping может быть очень полезным диагностическим инструментом. Она поможет быстро проверить, подключены ли вы к маршрутизатору или к интернету, и дает кое-какое представление о качестве этой связи.

41. NETHOGS

Если у вас медленный интернет, то вам, наверное, было бы интересно знать, сколько трафика использует какая-либо программа в Linux или какая программа потребляет всю скорость. Теперь это можно сделать с помощью утилиты nethogs. Для того чтобы задать сетевой интерфейс используйте опцию -i.

42. TRACEROUTE

Это усовершенствованная версия ping. Мы можем увидеть не только полный маршрут сетевых пакетов, но и доступность узла, а также время доставки этих пакетов на каждый из узлов.

Перенаправление ввода/вывода

Практически все ОС обладают механизмом перенаправления ввода/вывода. Linux не является исключением из этого правила. Обычно программы вводят текстовые данные с консоли (терминала) и выводят данные на консоль. При вводе под консолью подразумевается клавиатура, а при выводе - дисплей терминала. Клавиатура и дисплей - это, соответственно, стандартный ввод и вывод (stdin и stdout). Любой ввод/вывод можно интерпретировать как ввод из некоторого файла и вывод в файл. Работа с файлами производится через их дескрипторы. Для организации ввода/вывода в UNIX используются три файла: stdin (дескриптор 1), stdout (2) и stderr(3).
Символ > используется для перенаправления стандартного вывода в файл.
Пример:

$ cat > newfile.txt Стандартный ввод команды cat будет перенаправлен в файл newfile.txt, который будет создан после выполнения этой команды. Если файл с этим именем уже существует, то он будет перезаписан. Нажатие Ctrl + D остановит перенаправление и прерывает выполнение команды cat.
Символ < используется для переназначения стандартного ввода команды. Например, при выполнении команды cat < file.txt в качестве стандартного ввода будет использован файл file.txt, а не клавиатура.
Символ >> используется для присоединения данных в конец файла (append) стандартного вывода команды. Например, в отличие от случая с символом >, выполнение команды cat >> newfile.txt не перезапишет файл в случае его существования, а добавит данные в его конец.
Символ | используется для перенаправления стандартного вывода одной программы на стандартный ввод другой. Напрмер, ps -ax | grep httpd.

Конвейер

Однако как поступить, если вы хотите отсортировать данные, которые являются результатом работы какой-либо другой команды, например, **ls**?
Будем сортировать данные в обратном алфавитном порядке; это делается опцией -r команды **sort**. Если вы хотите перечислить файлы в текущем каталоге в обратном алфавитном порядке, один из способов сделать это будет таким. Применим сначала команду **ls**:
/home/larry/papers# ls
english-list
history-final
masters-thesis
notes
/home/larry/papers#

Теперь перенаправляем выход команды **ls** в файл с именем file-list
/home/larry/papers# ls > file-list
/home/larry/papers# sort -r file-list
notes
masters-thesis
history-final
english-list
/home/larry/papers#

Здесь выход команды **ls** сохранен в файле, а после этого этот файл был обработан командой **sort -r**. Однако этот путь является неизящным и требует использования временного файла для хранения выходных данных программы **ls**.

Решением в данной ситуации может служить создание *состыкованных команд* (pipelines). Стыковку осуществляет командная оболочка, которая stdout первой команды направляет на stdin второй команды. В данном случае мы хотим направить stdout команды **ls** на stdin команды **sort**. Для стыковки используется символ |, как это показано в следующем примере:

```
/home/larry/papers# ls | sort -r
notes
masters-thesis
history-final
english-list
/home/larry/papers#
```

Эта команда короче, чем совокупность команд, и её проще набирать.
Рассмотрим другой полезный пример. Команда

```
/home/larry/papers# ls /usr/bin
```

выдаёт длинный список файлов. Большая часть этого списка пролетает по экрану слишком быстро, чтобы содержимое этого списка можно было прочитать. Попробуем использовать команду more для того, чтобы выводить этот список частями:

```
/home/larry/papers# ls /usr/bin | more
```

Теперь можно этот список «перелистывать».
Можно пойти дальше и состыковать более двух команд. Рассмотрим команду **head**, которая является фильтром следующего свойства: она выводит первые строки из входного потока (в нашем случае на вход будет подан выход от нескольких состыкованных команд). Если мы хотим вывести на экран последнее по алфавиту имя файла в текущем каталоге, можно использовать следующую длинную команду:

```
/home/larry/papers# ls | sort -r | head -1 notes
/home/larry/papers\#
```

где команда **head -1** выводит на экран первую строку получаемого ей входного потока строк (в нашем случае поток состоит из данных от команды **ls**), отсортированных в обратном алфавитном порядке.

5.5.2 Оператор |

Особым вариантом перенаправления вывода является организация программного канала (иногда называет трубопроводом или конвейером). Для этого две или несколько команд, таких, что вывод предыдущей служит вводом для следующей, соединяются (или разделяются, если вам это больше нравится) символом вертикальной черты — "|". При этом стандартный выходной поток команды,

расположенной слева от символа `|`, направляется на стандартный ввод программы, расположенной справа от символа `|`. Например:

```
[user]$ cat myfile | grep Linux | wc -l
```

Эта строка означает, что вывод команды `cat`, т. е. текст из файла myfile, будет направлен на вход команды `grep`, которая выделит только строки, содержащие слово "Linux". Вывод команды `grep` будет, в свою очередь, направлен на вход команды `wc -l`, которая подсчитает число таких строк.

Программные каналы используются для того, чтобы скомбинировать несколько маленьких программ, каждая из которых выполняет только определенные преобразования над своим входным потоком, для создания обобщенной команды, результатом которой будет какое-то более сложное преобразование.

Надо отметить, что оболочка одновременно вызывает на выполнение все команды, включенные в конвейер, запуская для каждой из команд отдельный экземпляр оболочки, так что как только первая программа начинает что-либо выдавать в свой выходной поток, следующая команда начинает его обрабатывать. Точно так же каждая следующая команда выполняет свою операцию, ожидая данных от предыдущей команды и выдавая свои результаты на вход последующей. Если вы хотите, чтобы какая-то команда полностью завершилась до начала выполнения последующей, вы можете использовать в одной строке как символ конвейера `|`, так и точку с запятой **;**. Перед каждой точкой с запятой оболочка будет останавливаться и ожидать, пока завершится выполнение всех предыдущих команд, включенных в конвейер.

Статус выхода (логическое значение, возвращаемое после завершения работы программы) из канала совпадает со статусом выхода, возвращаемым последней командой конвейера. Перед первой командой конвейера можно поставить символ "!", тогда статус выхода из конвейера будет логическим отрицанием статуса выхода из последней команды. Оболочка ожидает завершения всех команд конвейера, прежде чем установить возвращаемое значение.

15.Процессы в ОС Linux.

Создание процесса

При создании процесса с ним ассоциируется 3 потока ввода/вывода

- stdin (стандартный поток ввода) — дескриптор 1
- stdout (стандартный поток вывода) — дескриптор 2
- stderr (стандартный поток ошибок) — дескриптор 3

- Процессы создаются другими процессами при помощи системного вызова fork()
- Между процессами устанавливаются отношения наследства

▪ «Осиротевшие» процессы наследуются процессом номер 1 — init
▪ Ядро поддерживает специальную таблицу процессов

Аттрибуты процесса

- Адресное пространство (в памяти)
    - Заголовок
    - Код процесса
    - Данные (data)
    - Куча (heap) — область памяти, предназначенная для выделения памяти динамическим переменным;
    - Стек (stack) — структура типа LIFO, предназначенная для вызовов подпрограмм и хранения некоторых переменных.
- Набор структур, содержащихся в ядре
    - Таблица распределения памяти
    - Текущий статус
    - Приоритет
    - Информация об используемых ресурсах
    - Информация об открытых файлах и сетевых портах
    - Запись о том, какие сигналы блокируются
    - Владелец процесса

Структура данных task_struct хранит данные для управления процессом:
▪ адресное пространство процесса
▪ состояние процесса:
    ▫ TASK_RUNNING (обозначается R)
    ▫ TASK_INTERRUPTIBLE (обозначается S)
    ▫ TASK_UNINTERRUPTIBLE (обозначается D)
    ▫ TASK_STOPPED (обозначается T)
    ▫ TASK_ZOMBIE (обозначается Z)
▪ приоритет процесса
▪ открытые файлы процесса
▪ обработчики сигналов процесса
▪ указатели на родительский процесс, дочерние и братские процессы

Просмотр информации о процессах
▪ ps - просмотр информации об исполняемых процессах;
▪ pstree - просмотр дерева исполняемых процессов;
▪ jobs - просмотр информации о заданиях.

Подробнее о **ps**

▪ ps — вывести информацию о процессах, связанных с текущим терминалом
PID — ID процесса, TTY — терминал, TIME — суммарное процессорное время
▪ ps -f — вывести более подробную информацию
UID — пользователь, PPID — ID родителя, C — уровень загрузки процессора, STIME — время запуска процесса
▪ ps -l — вывести еще более подробную информацию
▪ ps -e или ps -A — вывести информацию о всех процессах в системе
Опции фильтрации ps:
▪ u — по UID
▪ t — по терминалу
▪ p — по PID
▪ C — по командной строке
ps - C bash pgrep <имя команды> — получить список PID процессов по имени команды
pgrep -l <имя команды> — выводится не только PID, но и имя процесса
ps -aux — опции команды в BSD стиле
Примеры ps
Пример 1 $ ps -e $ ps -ef $ ps -eLf $ ps axms
Пример 2 $ ps -f -u www-data
Пример 3 $ ps -C apache2
Пример 4 $ ps aux --sort=-pcpu,+pmem
Пример 5 $ ps -o pid,uname,comm -C apache2

pstree [options] где options обозначают опции команды.
▪ -a - показать аргументы командной строки;
▪ -A - рисовать дерево ASCII символами;
▪ -c - не сжимать одинаковые поддеревья;
▪ -h - высвечивать текущий процесс и его предков;
▪ -H pid - высвечивать заданный процесс;
▪ -l - отображать длинные линии;
▪ -n - сортировать дочерние процессы по PID, а не по имени;
▪ -p - показывать PID процессов

jobs [OPTIONS] [JOBSPEC ...] где
- OPTIONS - опции команды;
- JOBSPEC - номер или имя задания.
Опции: ▪ -l ▪ -p ▪ -n ▪ -r ▪ -s
Примеры jobs
Пример 1 $ jobs
    [1]- Running nautilus &
    [2]+ Running jedit &
Пример 2 $ jobs -l

Пример 3 $ jobs -s

Планирование процессов

Операционная система Linux планирует работу родительских и дочерних процессов независимо друг от друга. Нет гарантии, что один процесс будет запущен раньше другого. и неизвестно, как долго один процесс будет выполняться, прежде чем Linux прервет его работу и передаст управление другому процессу. Linux лишь гарантирует, что любой процесс когда-нибудь получит свой квант времени: ни один процесс не окажется полностью лишенным доступа к процессору.

Ядро Linux применяет планировщик процессов для того, чтобы решить, какой процесс получит следующий квант времени.

Можно сообщить системе о том, что процесс не очень важен и должен выполняться с пониженным приоритетом. Это делается путем повышения *фактора уступчивости* процесса. По умолчанию у каждого процесса нулевой фактор уступчивости. Повышение этого значения свидетельствует о снижении приоритета процесса, и наоборот: процессы с низким (т.е. отрицательным) фактором уступчивости получают больше времени на выполнение.

Для запуска программы с ненулевым фактором уступчивости необходимо воспользоваться командой `nice -n`

Изменить фактор уступчивости выполняющегося процесса позволяет команда `renice`

Если требуется менять фактор уступчивости программным путем, надо использовать функцию `nice()`. Ее аргумент — это величина приращения, добавляемая к фактору уступчивости вызывающего процесса. В результате приоритет процесса снижается.

Только программа с привилегиями пользователя `root` может запускать процессы с отрицательным фактором уступчивости или понижать это значение у выполняющегося процесса. Это означает, что вызывать команды `nice` и `renice` с отрицательными аргументами можно, лишь войдя в систему как пользователь root, и только процесс, выполняемый от имени суперпользователя, может передавать функции `nice()` отрицательное значение. Таким образом, обычные пользователи не могут помешать работать процессам других пользователей и монополизировать системные ресурсы.

16.Командный интерпретатор Bash.

Командный интерпретатор, интерпретатор командной строки — компьютерная программа, часть операционной системы, обеспечивающая базовые возможности управления компьютером посредством интерактивного ввода команд через интерфейс командной строки или последовательного исполнения пакетных командных файлов.

Как правило его функции сводятся к предоставлению пользователю возможности запускать другие программы, может также содержать некоторые базовые команды ввода-вывода и свой простой скриптовый язык программирования.

Bash - это командный интерпретатор, с которым пользователь может работать в интерактивном режиме(Режим взаимодействия пользователя с компьютером, при котором каждый запрос пользователя вызывает немедленное ответное действие компьютера; обмен сообщениями между пользователем и компьютерной системой в режиме реального времени), используя консоль или терминал.

Командный интерпретатор Bash имеет свой **язык команд**.

Сценарий или **скрипт** (script) - программа, содержащая команды и инструкции, хранится в текстовом файле и исполняется интерпретатором.

Запуск командной оболочки:

1. Процесс login выполняет аутентификацию пользователя
2. Процесс login запускает командную оболочку пользователя, имя которой хранится в записи файла /etc/passwd

Список командных оболочек, доступных в системе, - /etc/shells

Файлы запуска:

Вызов в интерактивном режиме с регистрацией в системе:

- /etc/profile
- Файлы .bash_profile, .bash_login и .profile
  - устанавливаются цвета для вывода;
  - задается маска разрешений на доступ к файлам по умолчанию
  - устанавливаются переменные окружения для локализации;
  - конфигурируется настройки клавиатуры;
  - создается каталог временных файлов, имя которого задается в переменной TMPDIR;
  - корректирует переменную PATH

- исполняет сценарий из файла ~/.bashrc, который исполняет сценарий из файла /etc/bashrc ~/.bash_logout

Вызов интерактивной командной оболочки без регистрации в системе:

- /etc/bashrc
- ~/.bashrc

Неинтерактивное обращение к оболочке:

- переменная среды окружения BASH_ENV

Отладочный режим:

Когда дела идут не так, как планировалось, необходимо определить, из-за чего в скрипте возникли проблемы. В Bash для отладки предоставляются широкие возможности. Наиболее распространенным способом является запуск подоболочки с параметром -x, благодаря которому весь скрипт будет запущен в отладочном режиме. После того, как для каждой команды будут выполнены все необходимые подстановки и замены, но перед тем, как команда будет выполнена, в стандартный выходной поток будет выдана трассировка команды и все ее аргументы.

С помощью встроенной команды **set**, имеющейся в Bash, вы можете запускать в обычном режиме те части скрипта, относительно которых вы уверены, что они работают без ошибок, и отображать отладочную информацию только там, где есть подозрение на неправильную работу. Скажем, мы не уверены, что в примере commented-script1.sh будет делать команда **w**, поэтому мы можем окружить эту команду следующими отладочными командами:

```
set -x          # activate debugging from here
w
set +x          # stop debugging from here
```

17.Язык программирования BASH. Операторы условия, циклы. Переменные окружения.

Командный интерпретатор, интерпретатор командной строки — компьютерная программа, часть операционной системы, обеспечивающая базовые возможности управления компьютером посредством интерактивного ввода команд через интерфейс командной строки или последовательного исполнения пакетных командных файлов.

Bash - это командный интерпретатор, с которым пользователь может работать в интерактивном режиме, используя консоль или терминал.

Командный интерпретатор Bash имеет свой **язык команд**.

Возможности:

Арифметика

В bash есть рудиментарная поддержка целочисленной арифметики. Арифметическое выражение (возможно, содержащее переменные) можно взять в «толстые скобки» $((и)).

Массивы

Каждая переменная в bash может иметь несколько значений, индексируемых целыми неотрицательными числами.

Условия

Каждая выполненная команда оболочки может завершиться как успехом, так и ошибкой. В случае успеха в переменную автоматически записывается число 0. В случае ошибки — какое-нибудь ненулевое число.

Циклы

Из циклов чаще всего применяется «простой for»

Также изредка бывает полезен цикл while:

Функции

Функции в bash — механизм определения новых команд.

Дескрипторы

Для общения со внешним миром процесс использует каналы, называемые *дескрипторами*. Каждый дескриптор идентифицируется целым неотрицательным числом.

Асинхронный запуск команды

Обычно при запуске любой команды bash дожидается её завершения (если команда внешняя, то он на время выполнения команды засыпает). Это называется *синхронным* запуском.

Любую команду можно запустить в *асинхронном режиме*, поставив в самом конце команды символ &. В этом случае команда *всегда* запускается как новый процесс (являющийся потомком bash), при этом bash не входит в спящий режим.

18.Программирование в ОС Linux: инструментарий и менеджеры пакетов.

Инструментарий: Операционная система Linux (GNU/Linux) с текстовым редактором (vim, nvi; emacs); компилятор (gcc) и компоновщик (gnu 1d); дополнительные утилиты (make).

Debian – основанные менеджеры пакетов

Менеджер пакетов Dpkg

Ubuntu и Debian считаются одним из наиболее широко используемых потребительских операционных систем на основе Linux, имеющихся сегодня на рынке. Их менеджеры пакетов являются общими, с системой управления пакетами нижайшего уровня «Dpkg», сокращенно от «Debian Package». Это скелет программного обеспечения для управления пакетами, с инструментами для установки, удаления и сборки пакетов.

В Dpkg не хватает более расширенные возможности – функциональные возможности, такие как загрузка пакетов из Интернета, или установка зависимостей автоматически не возможно через DPKG. Будучи в состоянии сделать это из Интернета очень полезна, так как она позволяет пользователям добавлять репозитории пакетов, что значительно увеличивает выбор программного обеспечения, которое может быть легко установлено в системе. Оно также может значительно упростить процесс установки программного обеспечения, будучи в состоянии легко найти и установить пакет только с одной командой.

Менеджер пакетов APT

Это где интерфейсы, такие как apt и aptitude вступают в игру. APT, сокращенно от Advanced Package Tool, гораздо более продвинутый в функциональности по сравнению с dpkg. Он также может устанавливать, удалять и собирать пакеты – однако его функциональность идет гораздо дальше. APT может обновить свои пакеты, установить зависимости автоматически, а также загрузить пакеты из интернета. Это один из наиболее распространенных менеджеров пакетов, установленных на современных дистрибутивах, с предустановленными на Ubuntu, Debian и большинстве других операционных систем на основе Debian.

Менеджер пакетов Aptitude

Aptitude очень похож на APT, предлагая большую часть той же функциональности. Но он может предложить несколько дополнительных функций, таких как безопасные обновления, позволяющие пользователям обновлять свои пакеты, не удаляя существующие пакеты из системы. Также доступно удержание пакетов, что предотвращает автоматическое обновление некоторых пакетов.

Оба этих менеджера пакетов фактически используют dpkg для основных операций и используют только свое собственное программное обеспечение для загрузки пакетов и управления ими.

Менеджеры пакетов RedHat Enterprise Linux (RHEL)

Менеджер пакетов RPM

Redhat и CentOS являются одним из наиболее широко используемых серверных операционных систем на серверах. Основное программное обеспечение для управления пакетами в этих системах-RPM, сокращенно от Red Hat Package Manager. Этот менеджер пакетов также выполняет основные операции, такие как установка и удаление пакетов, и,

как dpkg, также не может управлять пакетами или устанавливать их непосредственно из интернета.

Менеджер пакетов YUM

Как и операционные системы на основе Debian, операционные системы RHEL также имеют собственное программное обеспечение для управления пакетами. YUM, короткий для Yellow Dog Updater, является самым популярным выбором в качестве интерфейса RPM. Он открывает гораздо больше возможностей для RPM-файлов через репозитории, отслеживая то, что установлено в системе, оптимизированное обновление и многое другое. RHEL эквивалентен менеджеру пакетов APT.

Менеджер пакетов DNF

DNF, сокращенно от Dandified Packaging Tool, – это более модернизированная и усовершенствованная версия менеджер пакетов yum – вбирая в себя черты yum, при одновременном повышении производительности и использования ресурсов. На данный момент только Fedora использовала эту версию следующего поколения YUM, но, надеюсь, мы увидим, что она будет распространятся на другие операционные системы в будущем.

Существует несколько других инструментов управления пакетами, доступных для систем на основе RPM, таких как up2date, urpmi и ZYpp, однако они не так широко используются, как YUM или DNF.

_____ из дока на коллоквиум_____

Термин программное обеспечение с открытым исходным кодом — маркетинговое название бесплатных программ, создаваемых сообществом Open Source Initiative (OSI)
Самые распространенные архиваторы с открытым кодом:

Архиватор	**Примечание**
`tar`	Самый популярный
`cpio`	Служит для работы с форматом RPM
`ar`	Служит для создания пакетных файлов Debian

Расширение	**Тип**
.tar	**Архив TAR, несжатый**
.tar.gz, .tgz	**Архив TAR, сжатый с помощью gzip**
.tar.bz2	**Архив TAR, сжатый с помощью bzip2**

.tar.Z, .taz	**Архив TAR, сжатый с помощью команды UNIX compress**
.ar, .a	**Архив AR, используется преимущественно при разработке программного обеспечения**
.cpio	**Архив CPIO, несжатый**

Если вы не можете точно идентифицировать файл, воспользуйтесь командой file. Она позволит вам идентифицировать файл, если его имя ни о чем вам не говорит

```
$ file foo.x
foo.x : gzip compressed data, from UNIX, max compression
$ file -z foo.x
foo.x: tar archive (gzip compressed data, from UNIX, max
compression)
```

Содержимое архивных файлов можно отслеживать посредством оглавления, доступ к которому довольно легко получить, добавив флаг –t к любому из уже упоминавшихся архиваторов.

```
$ tar -tzvf data.tar.gz
drwxr-xr-x root/root 0 2001-10-01 07:53:19 ./
drwxr-xr-x root/root 0 2001-10-01 07:53:15 ./usr/
drwxr-xr-x root/root 0 2001-10-01 07:53:18 ./usr/bin/
-rwsr-xr-x root/root 22460 2001-10-01 07:53:18 ./usr/bin/crontab
drwxr-xr-x root/root 0 2001-10-01 07:53:18 ./usr/sbin/
-rwxr-xr-x root/root 25116 2001-10-01 07:53:18 ./usr/sbin/cron
```

В табл. приведены основные команды, используемые для просмотра содержимого архивов различных форматов

Формат	Команда	Примечание
TAR	tar -tvf filename	—
Архив TAR, сжатый с помощью gzip	tar -tzvf filename	—
Архив TAR, сжатый с помощью bzip2	tar -tjvf filename	—
Архив CPIO	cpio -tv < filename	CPIO использует stdin и stdout как двоичные потоки

Рассмотрим строку разрешений. Она состоит из десяти символов. Первый символ указывает на тип файла, а остальные три группы из трех символов резюмируют разрешение владельца файла, разрешения членов группы и разрешения других лиц соответственно. Тип файла определяется одним символом. В табл приведены допустимые значения этого символа и пояснения к ним.

Символ	Пояснение	Примечание
-	Обычный файл	Сюда входят текстовые файлы, файлы данных, исполняемые файлы и т. д.
d	Каталог	
c	Символьное устройство	Особый файл, используемый для взаимодействия с драйвером символьного устройства. Традиционно эти файлы сосредоточены в каталоге /dev; в архивах они обычно отсутствуют
b	Блочное устройство	Особый файл, применяемый для взаимодействия с драйвером блочного устройства. По традиции эти файлы сосредоточены в каталоге /dev; в архивах они обычно отсутствуют
l	Символическая ссылка	Имя файла, указывающее на другое имя файла. Файл, на который оно указывает, может располагаться в другой файловой системе либо вообще не существовать

Примеры разрешений

Строка разрешения	Разрешение на исполнение	Эффективный идентификатор пользователя	Эффективный идентификатор группы
-rwxr-xr-x	Файл может исполняться всеми пользователями	Текущий пользователь	Текущий пользователь
-rw-r-xr-x	Файл может исполняться всеми членами файловой группы за исключением владельца; файл может исполняться всеми пользователями за исключением владельца	Текущий пользователь	Текущий пользователь
-rwsr-xr-x	Файл может исполняться всеми пользователями	Владелец файла	Текущий пользователь
-rwSr-xr-x	Файл может исполняться всеми пользователями за исключением владельца	Владелец файла	Текущий пользователь
-rwxr-sr-x	Файл может исполняться всеми пользователями	Текущий пользователь	Владелец группы
-rwsr-sr-x	Файл может исполняться всеми пользователями	Владелец файла	Владелец группы
-rwsr-Sr-x	Файл может исполняться всеми пользователями, включая владельца, за исключением членов файловой группы	Владелец файла	Владелец группы

Команды извлечения файлов из архива

Формат	Команда
TAR	tar -xf filename
Архив TAR, сжатый с помощью gzip	tar -xzf filename
Архив TAR, сжатый с помощью bzip2	tar -xjf filename
Архив CPIO	cpio -i -d < filename
Архив AR	ar x filename

CPIO полностью сохраняют путь по отношению к корневому каталогу. Пример команды:

```
cpio -t < foo.cpio
```

```
/etc/hosts
```

Пакетные менеджеры

Operating System	Format	Tool(s)
Debian	`.deb`	`apt`, `apt-cache`, `apt-get`, `dpkg`
Ubuntu	`.deb`	`apt`, `apt-cache`, `apt-get`, `dpkg`
CentOS	`.rpm`	`yum`
Fedora	`.rpm`	`dnf`
FreeBSD	Ports, `.txz`	`make`, `pkg`

Основные функции пакетных менеджеров

- установка в системе новых программ;
- удаление программ из системы;
- проверка;
- модернизация версий установленных в системе программ;
- запрос установленного программного обеспечения;
- извлечение.

Безопасность и пакеты

Вредоносное программное обеспечение:

- Вирусы;
- Шпионские программы;
- Деструктивное ПО.

Для обеспечения должного уровня безопасности Linux не позволяет

непривилегированным пользователям устанавливать программы.

Основная форма проверки подлинности пакетов — функция хеширования.

При наличии хеша из доверенного источника:

- шансы получить модифицированный файл с идентичным хешем очень невелики;
- вероятность того, что злоумышленник может взять файл, модифицировать его по своему усмотрению и сгенерировать идентичный хеш, бесконечно мала.

Создание хеша:

```
$ md5sum foo.tar bar.tar
```

Верификация хеша:

```
$ md5sum foo.tar bar.tar > md5.sums
```

```
$ md5sum  — -check md5.sums
```

Цифровая подпись

```
$ rpm –checksig *.rpm
```

Если проверка подлинности невозможна

- Используйте исходный код для сборки своих приложений.
- Проверяйте сценарии, исполняемые в процессе установки.
- Проверяйте содержимое.

Состав пакетного файла

Архив с файлами для установки;

Сценарии, исполнямые во время установки и удаления пакета;

Сведения о зависимостях;

Текстовые данные о самом пакете.

Типы сценариев установки

- Прединсталляционные
- Постинсталляционные
- Преддеинсталляционные
- Постдеинсталляционные

Как выполнить проверку пакетов

Запрос	RPM	Debian
Базовые сведения	rpm -qpi имя_файла	dpkg -s имя_файла
Список файлов для установки	rpm -qpl имя_файла	dpkg -L имя_файла
Дамп сценариев установки/удаления	rpm -qp -scripts имя_файла	dpkg -e
Проверка аутентификационных сведений	rpm --checksig имя_файла	Отсутствует
Показать пакеты, необходимые для данного пакета	rpm -qp --requires имя_файла	dpkg -I
Показать, какой пакет представляет этот файл (например, его имя и версия, отображаемые в базе данных)	rpm -qp --provides имя_файла	dpkg -I

19.Программирование в ОС Linux: сборка.

Инструменты для сборки

Утилита **make** — первый инструмент с поддержкой итеративного процесса сборки ПО.

Варианты:

imake;

Сборочный инструментарий GNU;

Редактор связей — ld

LD создавался для обеспечения максимально возможного контроля за процессом линковки и совместимости с другими линкерами. В результате в LD большое число опций, управляющих поведением линкера.

Библиотеки:

- Статические — .a
- Динамические — .so

*.a - **a**rchive library:

содержит библиотеку функций и заголовков, на которые может ссылаться исходный файл C / C ++

.so - position independent code shared library

Общая библиотека, загружаемая программами C и C ++ при их запуске; содержит функции и другую общую программную логику;

```
(это что то типо dll библиотеки)
```

Для создания используется архиватор ar.

Этапы сборки GNU программы

```
$ ./configure
$ make
$ make install
```

Сценарий configure

На этапе конфигурирования осуществляются все мелкие операции, с которыми нельзя справиться только с помощью файла Makefile

Параметр	**Применение**
--prefix	Определяет корневой каталог установки, обычно это /usr/local
--bindir	Определяет местоположение файлов, которые в противном случае будут установлены в ${prefix}/bin. Данный каталог предназначен для исполняемых файлов, которые будут запускаться регулярными пользователями

--sbindir	Определяет местоположение файлов, которые в противном случае будут установлены в ${prefix}/sbin. Данный каталог предназначен для исполняемых файлов, которые будут запускаться администраторами или системными демонами
--datadir	Определяет местоположение служебных файлов с данными; по умолчанию ${prefix}/share
--libdir	Определяет местоположение совместно используемых библиотек; по умолчанию ${prefix}/lib
--mandir	Определяет местоположение установки страниц руководства man; по умолчанию ${prefix}/man
--program- prefix	Добавляет префикс к исполняемым файлам, расположенным в ${bindir} и ${sbindir). Иногда используется для установки нескольких версий одной программы, так как позволяет присваивать уникальный префикс каждой версии
--program- suffix	Аналог -program-prefix за тем исключением, что добавляет к программе суффикс

Чтобы увидеть доступные команды:

```
$ ./configure –help
```

Этап сборки: make

Этап сборки начинается с ввода команды make без аргументов. После этого выполняется сборка всех объектных файлов и связывание их с одной или несколькими программами

Цель	**Применение**
all	Цель по умолчанию; осуществляет сборку всех библиотек и исполняемых файлов
clean	Удаляет объектные файлы и библиотеки, сборка которых осуществлялась целью all
distclean	Удаляет все файлы, созданные во время этапа конфигурирования и сборки.В идеале должны остаться только те файлы, которые имелись в исходном архиве tar. После сборки цели distclean файл Makefile будет отсутствовать, и вам придется вновь использовать configure перед выполнением пересборки

mostyclean	Аналог clean за тем исключением, что не удаляются избранные библиотеки, включая те из них, что определены разработчиками как редко изменяемые и не нуждающиеся в пересборке
`install`	Устанавливает программы в каталоги, определенные на этапе конфигурирования. Следует проявлять бдительность, поскольку цель uninstall иногда может отсутствовать
`uninstall`	Если присутствует, то служит для удаления программ, установленных целью install

Этап установки: make install

Если вы решили установить программу для общего пользования, то конфигурации по умолчанию будет достаточно. Для этого вам потребуются привилегии корневого пользователя root

Безопасный каталог для установки по умолчанию — `/usr/local/bin`

Установка в домашнем каталоге пользователя, например:

```
$ ./configure –prefix ~/usr
```

20.Архитектура Android. Компоненты и безопасность Android-приложений.

Архитектура Android:

Уровень приложений (Applications)

В состав Android входит комплект базовых приложений: клиенты электронной почты и SMS, календарь, различные карты, браузер, программа для управления контактами и много другое. Все приложения, запускаемые на платформе Android написаны на языке Java.

Уровень каркаса приложений (Application Framework)

Android позволяет использовать всю мощь API, используемого в приложениях ядра. Архитектура построена таким образом, что любое приложение может использовать уже реализованные возможности другого приложения при условии, что последнее откроет доступ на использование своей функциональности. Таким образом, архитектура реализует принцип многократного использования компонентов ОС и приложений.

Основой всех приложений является набор систем и служб:
1. Система представлений (**View System**) – это богатый набор представлений с расширяемой функциональностью, который служит для построения внешнего вида приложений, включающий такие компоненты, как списки, таблицы, поля ввода, кнопки и т.п.
2. Контент-провайдеры (**Content Providers**) – это службы, которые позволяют приложениям получать доступ к данным других приложений, а также предоставлять доступ к своим данным.
3. Менеджер ресурсов (**Resource Manager**) предназначен для доступа к строковым, графическим и другим типам ресурсов.
4. Менеджер извещений (**Notification Manager**) позволяет любому приложению отображать пользовательские уведомления в строке статуса.
5. Менеджер действий (**Activity Manager**) управляет жизненным циклом приложений и предоставляет систему навигации по истории работы с действиями.

Уровень библиотек (Libraries)

Платформа Android включает набор C/C++ библиотек, используемых различными компонентами ОС. Для разработчиков доступ к функциям этих библиотек реализован через использование Application Framework. Ниже представлены некоторые из них:
1. **System C library** — BSD-реализация стандартной системной библиотеки C (libc) для встраиваемых устройств, основанных на Linux.
2. **Media Libraries** – библиотеки, основанные на PacketVideo‘s OpenCORE, предназначенные для поддержки проигрывания и записи популярных аудио- и видео-форматов (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG и т.п.).
3. **Surface Manager** – менеджер поверхностей управляет доступом к подсистеме отображения 2D- и 3D- графических слоев.
4. **LibWebCore** – современный движок web-браузера, который предоставляет всю мощь встроенного Android-браузера.
5. **SGL** – движок для работы с 2D-графикой.
6. **3D libraries** – движок для работы с 3D-графикой, основанный на OpenGL ES 1.0 API.
7. **FreeType** – библиотека, предназначенная для работы со шрифтами.
8. **SQLite** – мощный легковесный движок для работы с реляционными БД.

Уровень среды исполнения (Android Runtime)

В состав Android входит набор библиотек ядра, которые предоставляют большую часть функциональности библиотек ядра языка Java.

Платформа использует оптимизированную, регистр-ориентированную виртуальную машину Dalvik(или ART), в отличии от нее стандартная виртуальная машина Java –

стек-ориентированная. Каждое приложение запускается в своем собственном процессе, со своим собственным экземпляром виртуальной машины. Dalvik использует формат Dalvik Executable (*.dex), оптимизированный для минимального использования памяти приложением. Это обеспечивается такими базовыми функциями ядра Linux, как организация поточной обработки и низкоуровневое управление памятью. Байт-код Java, на котором написаны ваши приложения, компилируются в dex-формат при помощи утилиты dx, входящей в состав SDK.

Уровень ядра Linux (Linux Kernel)

Android основан на ОС Linux версии 2.6 (в данный момент - в основном на 3.10), тем самым платформе доступны системные службы ядра, такие как управление памятью и процессами, обеспечение безопасности, работа с сетью и драйверами. Также ядро служит слоем абстракции между аппаратным и программным обеспечением.

Четыре типа компонентов:

Activity

Activity представляет собой один экран с пользовательским интерфейсом. Например, в приложении для работы с электронной почтой одна Activity может служить для отображения списка новых сообщений, другая – для составления сообщения и третья Activity – для чтения сообщений. Несмотря на то что Activity совместно формируют связное взаимодействие пользователя с приложением по работе с электронной почтой, каждая из них не зависит от других операций. Любые из этих Activity могут быть запущены другим приложением.

Service

Service представляет собой компонент, который работает в фоновом режиме и выполняет длительные операции, связанные с работой удаленных процессов. Service не имеет пользовательского интерфейса. Например, он может воспроизводить музыку в фоновом режиме, пока пользователь работает в другом приложении, или же он может получать данные по сети, не блокируя взаимодействие пользователя с операцией. Service может быть запущен другим компонентом, который затем будут взаимодействовать с ним, – например операцией.

Content provider

Content provider управляет общим набором данных приложения. Данные можно хранить в файловой системе, базе данных SQLite, в Интернете или любом другом постоянном месте хранения, к которому у вашего приложения имеется доступ. Посредством поставщика

контента другие приложения могут запрашивать или даже изменять данные (если поставщик контента позволяет делать это).
ContentProvider должен реализовывать стандартный набор API-интерфейсов, с помощью которых другие приложения будут выполнять транзакции.

Broadcast receiver

Broadcast receiver представляет собой компонент, который реагирует на объявления, распространяемые по всей системе. Многие из этих объявлений рассылает система — например объявление о том, что экран выключился, аккумулятор разряжен. Объявления также могут рассылаться приложениями, — например, чтобы сообщить другим приложениям о том, что какие-то данные были загружены на устройство и теперь готовы для использования. Несмотря на то что приемники широковещательных сообщений не имеют пользовательского интерфейса, они могут создавать уведомления в строке состояния, чтобы предупредить пользователя о событии "рассылка объявления". Однако чаще всего они являются просто "шлюзом" для других компонентов и предназначены для выполнения минимального объема работы. Например , они могут инициировать выполнение службой определенных действий при возникновении события.
Приемник широковещательных сообщений относится к подклассу класса BroadcastReceiver , а каждое такое сообщение предоставляется как объект Intent.

Безопасность в ОС Android

Классический Unix — модель безопасности в классическом Unix основана на системе UID/GID.

1. Каждый UID (user ID) соответствует своему пользователю.
2. Чтобы обеспечить общий доступ, пользователи объединяются в группы.
3. Приложения, выполняемые пользователем, получают доступ ко всем его данным. Права пользователя и есть права программ, запущенных от его имени.
4. Исключение из ограничений доступа — UID 0 — или root пользователь.

В современном Linux модель безопасности на основе UID/GID была расширена и обобщена:

1. Capabilities, позволяющие «получить часть root-прав»,
2. Мандатное управление доступом (mandatory access control, MAC) и подсистема SELinux

Android позволяет вручную устанавливать произвольные приложения из доверенных источников (Google play store) или APK-файлов (это называют sideloading).

Android использует для ограничения доступа существующий механизм UID/GID — разные UID соответствуют разным приложениям.
Процессы разных приложений запускаются с разными UID. На уровне ядра приложения защищены и изолированы друг от друга и не имеют доступа к системе и данным пользователя. Это образует **песочницу** (Application Sandbox)

Чтобы получить доступ к пользовательским данным, приложение должно получить от пользователя **разрешение** (permission):

- Некоторые из разрешений — в виде GID. Например, получение разрешения *ACCESS_FM_RADIO* помещает приложение в группу *media*, что позволяет ему получить доступ к файлу */dev/fm*.
- Другие существуют в виде записей в файле packages.xml и проверяются другими компонентами системы при к API через Binder.

21.Структура Android-приложения.

По умолчанию Android Studio отображает проект в представлении Android (Android view). Данное представление не отображает существующую файловую иерархии на диске, но организовано по модулям и типам файлов, чтобы упростить навигацию между ключевыми исходными файлами проекта. Скрывая для этого определённые виды файлов и каталогов, которые не часто используются. Ряд структурных элементов проекта, которые соответствуют файловой структуре на диске включают следующие:

- Вывод всех конфигурационных файлов для сборки в верхнеуровневой группе Gradle Script.
- Вывод всех файлов манифестов для каждого модуля на уровне группы модуля. (для тех случаев, когда создаются отдельные файлы манифестов для сборки и платформ).

Каждый модуль приложения Android включает следующие группы файлов:

- manifests — содержит файл AndroidManifest.xml. На данный момент этот каталог практически не используется, файл манифеста находится в корне.
- assets - содержит файлы эссетов, например, шрифты.
- java — содержит исходные файлы Java, разделенные по пакетам, включая код тестов JUnit.
- res — содержит все ресурсы, такие как xml разметка, строки UI, изображения, размещаемые в соответствующих подкаталогах:
- layout - в данной папке содержатся xml-файлы, которые описывают внешний вид форм и их элементов;
- values - содержит XML файлы,которые определяют простые значения, таких ресурсов как, строки, числа, цвета, темы, стили, которые можно использовать в данном проекте.

Чтобы просмотреть файловую структуру проекта, включая все скрытые файлы в представлении Android (Android View), необходимо переключиться в представление Project в выпадающем меню окна Project.

В представлении Project (Project view) можем увидеть больше файлов и каталогов, наиболее важные из них:
build/ — содержит сгенерированные файлы сборки.
libs/ — содержит библиотеки
src/ — содержит код и файлы ресурсов модуля, распределенные по подкаталогам.
test/ - содержит код юнит-тестов, которые выполняются локально на JVM.
androidTest/ — содержит код тестов, которые выполняются на устройстве Android или эмуляторе.
main/ — содержит исходный код и ресурсы. Именно в этой папке размещаются все классы, создаваемые в процессе разработки приложения.

AndroidManifest.xml — описывается структуру приложения и каждого из его компонентов.
java/ — исходный код на java
res/ — содержит ресурсы приложение, такие как изображения, файлы разметки, файлы локализации и др.

Рассмотрим файл AndroidManifest.xml - файл в формате xml, который описывает основные свойства проекта, разрешение на использование ресурсов устройства и др.Это файл манифеста проекта.

В сгенерированном файле apk может содержаться только один файл AndroidManifest.xml, но проект Android Studio может содержать несколько файлов манифестов: файл манифеста проекта, манифест сборки и манифесты импортированных библиотек. Поэтому при создании приложения Gradle build объединяет все файлы манифеста в один файл манифеста, который упаковывается в файл apk приложения.

В Android Studio используется инструмент слияния манифестов, который объединяет все элементы XML из каждого файла, следуя правилам слияния. В результате слияния получается файл манифеста, содержащий все требуемые элементы.

Основная информация в манифесте:

- Имя Java пакета приложения
- Описание компонентов приложения
- Определение процессов
- Объявление полномочий, которыми должно обладать приложение для доступа к защищенным частям API и взаимодействия с другими приложениями
- Объявление полномочий, которыми должны обладать другие приложения для взаимодействия с компонентами данного
- Список вспомогательных классов

- Определение минимального уровня Android API для приложения
- Список библиотек связанных с приложением

ПРИМЕР МАНИФЕСТА

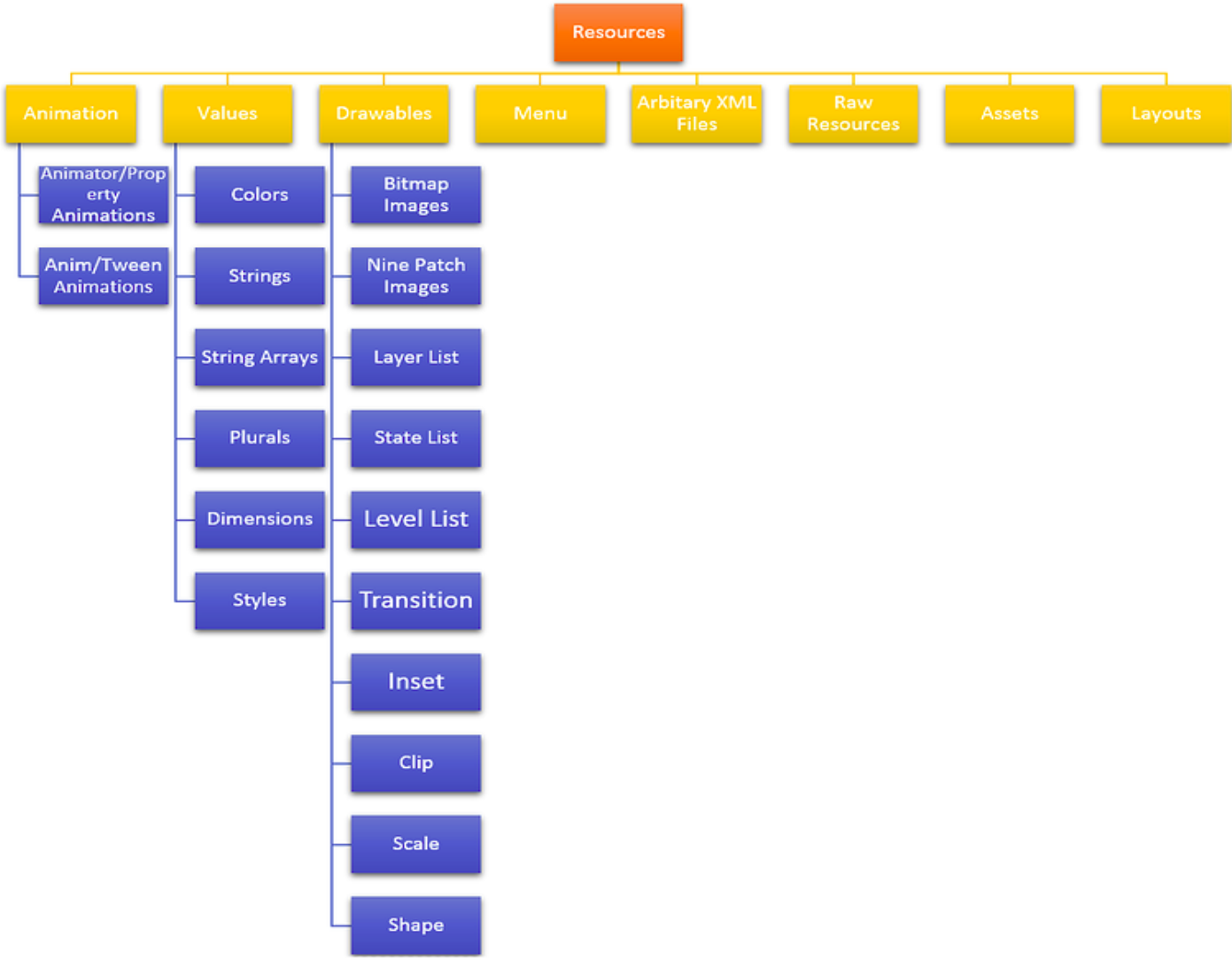
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.pupkin.myapplication">
   <application
       android:allowBackup="true"
       android:icon="@mipmap/ic_launcher"
       android:label="@string/app_name"
       android:roundIcon="@mipmap/ic_launcher_round"
       android:supportsRtl="true"
       android:theme="@style/AppTheme">
       <activity android:name=".MainActivity">
           <intent-filter>
               <action android:name="android.intent.action.MAIN" />
               <category
android:name="android.intent.category.LAUNCHER" />
           </intent-filter>
       </activity>
       <activity android:name=".ActivityTwo"></activity>
   </application>
</manifest>
```

Android:name - определяет имя класса, который задает активность.
Android:label -- определяет видимый пользователю заголовок активности, если он отличается от общего заголовка приложения. Задается как ссылка на строковый ресурс.
Android:icon - - определяет иконку для приложения целиком, а также иконку по умолчанию для компонентов приложения.
Android:allowBackup - определяет разрешение для приложения участвовать в резервном копировании и восстановлении.
На самом деле у элементов <application> и <**activity>** гораздо больше атрибутов.

Чаще всего, при создании приложения приходится иметь дело с папками src, res/layout и res/values, т.к. там находятся основные файлы проекта.

22.Ресурсы Android-приложения.

- res/
  - values/
  - drawable/
  - color/
  - layout/
  - menu/
  - anim/
  - xml/
  - raw/
- assets/



Идентификаторы ресурсов

Присвоение идентификатора — @+id/editText

Обращение к ресурсу — R.id.editText

Создание ресурса item заранее

```
<resources>
    <item type="id" name="text"/>
</resources>
<TextView android:id="@id/textView">
    </TextView>
```

Строковые ресурсы

Строковые ресурсы помогают упростить процесс создания локализованных версий.

Строковые ресурсы обозначаются тегом <string>.

Системные строковые ресурсы

```
android:text="@android:string/cancel"
```

Булевые ресурсы

Определение булевого ресурса

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <bool name="autostart">true</bool>
    <bool name="sound">no</bool>
</resources>
```

Получить значение через код:

```
Resources res = getResources();
boolean autostartSetting = res.getBoolean(R.bool.autostart);
```

В макете для булевых атрибутов

```
<ImageView
    android:adjustViewBounds="@bool/adjust_view_bounds" />
```

Числовые ресурсы

Определение числового ресурса

```
     <integer name="max_speed">75</integer>
```

Получить значение через код:

```
Resources res = getResources();
int maxSpeed = res.getInteger(R.integer.max_speed);
```

Ресурсы меню

- Хорошая практика — создавать меню в формате xml, используя отдельные файлы.
- Меню, описанное в формате XML, загружается в виде программного объекта с помощью метода inflate, принадлежащего сервису MenuInflater.

В XML-файле меню есть три элемента:

- <menu> — корневой элемент файла меню;
- <group> — контейнерный элемент, определяющий группу меню;
- <item> — элемент, определяющий пункт меню:
    - атрибуты item: id, title, icon

Ресурсы разметки

пример кода разметки: `setContentView(R.layout.main)`

Пример xml-файла с разметкой:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
      ...
      >
<TextView
      …
      />
</LinearLayout>
```

Цветовые ресурсы

Для работы со цветом используется тег <color>:

- #RGB;
- #RRGGBB;
- #ARGB;
- #AARRGGBB;

Предопределенные названия цветов:

- background_light
- black
- holo_blue_bright
- ….

Обычно используется файл colors.xml в подкаталоге res/values:

```
      <color name="green">#FF00FF00</color>
```

Программное использование цветовых ресурсов:

```
int myRedColor = activity.getResourses.getColor(R.color.red); //
получаем значение красного цвета
```

Использование цветовых ресурсов в xml-файлах:

```
<TextView
      …
    android:color="@color/red" />
```

Ресурсы размеров

Использование единиц размеров:

```
<dimen name="in_dp">5dp</dimen>
<dimen name="in_sp">100sp</dimen>
```

Использование ресурсов размеров:

```
float dimen =
activity.getResources().getDimension(R.dimen.in_pixels);
```

Использование ресурсов размеров в xml-файлах:

```
<TextView
      ...
    android:textSize="@dimen/in_dp"/>
```

Ресурсы визуальных стилей и тем

- Визуальные стили и темы используются для хранения цветовых значений и шрифтов — styles.xml.
- Вместо описания каждого стиля можно использовать ссылки, предоставляемые Android.
- Для создания стилей используется тег <style>:

```
<style name="StyleName">
     <item name="attributeName">value</item>
</style>
```

Создание ссылок на ресурсы:

```
android:textColor="?android:textColor"
android:background="?colorPrimary"
```

Ресурсы /res/drawable

res/drawable

- android:shape: rectangle, oval, line, ring;
- <corners>;
- <gradient>: Linear, Radial и Sweep;
- <size>;
- <solid>.

Разрешения экрана — drawable-hdpi, drawable-mdpi, drawable-ldpi и др.
Размеры экрана — drawable-normal, drawable-large и др.
Ссылка на изображение в xml-файле:

```
<Button
     ...
     android:background="@drawable/cat"
</Button>
```

Программное использование:

```
// вызываем getDrawable для получения изображения
BitmapDrawable bd =
activity.getResources().getDrawable(R.drawable.cat);
// Затем можно использовать полученный объект, чтобы установить фон
button.setBackgroundDrawable(bd);
// или можно установить фон непосредственно по идентификатору ресурса
button.setBackgroundResource(R.drawable.icon);
```

Получение имени ресурса:

```
getResources().getIdentifier("image_name","drawable",getPackageName()
)
```

Получение идентификатора ресурса:

```
String mDrawableName = "cat1"; // файл cat1.png в папке drawable
```

```
int resID = getResources().getIdentifier(mDrawableName , "drawable",
getPackageName());
```

Ресурсы отрисовываемых цветов

Создание цветного прямоугольника — тег <drawable> в файле /res.values:

```
<drawable name="black_rectangle">#000000</drawable>
```

Использование в xml-шаблонах:

```
<TextView
    ...
    android:background="@drawable/white_rectangle" />
```

Программное использование:

```
// Получение отрисовываемого объекта
ColorDrawable whiteDrawable =
(ColorDrawable)activity.getResources().getDrawable(R.drawable.white_r
ectangle);
// установление его в качестве фона для текстового вида
textView.setBackground(whiteDrawable);
```

Ресурсы mipmap

- Mipmap — это замена ресурсам Drawable для значков приложения.
- Основная идея — при компиляции неиспользуемые drawable-ресурсы могут быть удалены в целях оптимизации.
- Перенос значков приложения в новые папки с разными разрешениями позволяет избежать потенциальной проблемы с удалением нужных файлов.
- Подготовить значки и расположить их в папках mipmap-mdpi, mipmap-hdpi и т.д.

Ресурсы анимации

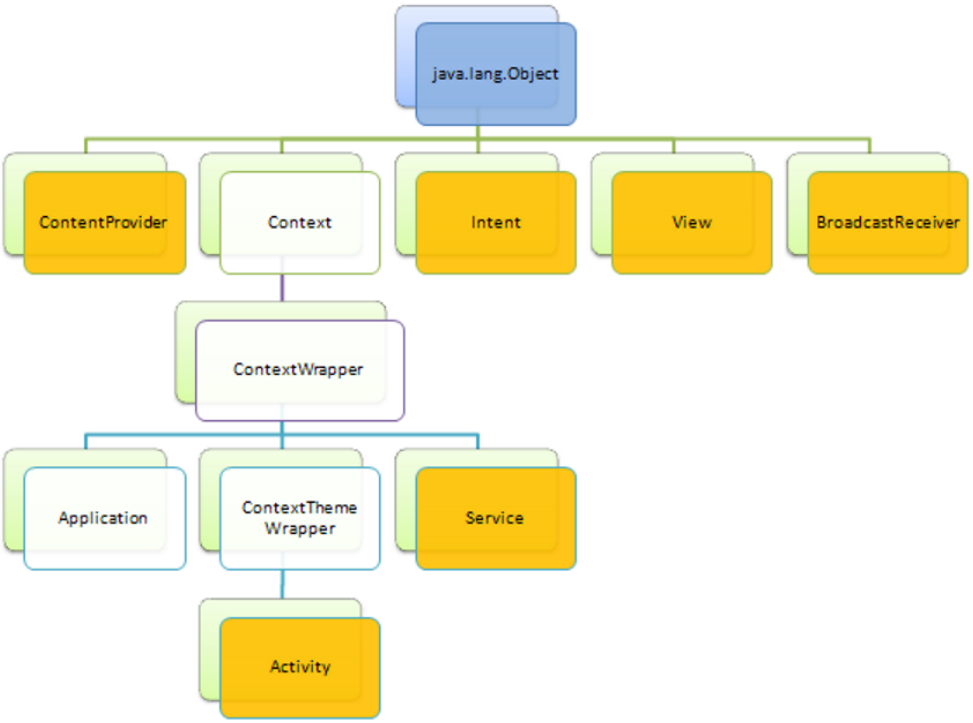
- Анимация свойств (Property Animation)
- Анимация представления (View Animation):
    a. **Tween Animation** — основана на расчете промежуточных кадров и может использоваться для поворота, перемещения, растягивания, затемнения элементов.
        i. TranslateAnimation
        ii. ScaleAnimation
        iii. RotateAnimation
        iv. AlphaAnimation
    b. **Frame Animation** — пошаговая анимация, т.е. последовательный вывод заранее подготовленных изображений.

Каждый экземпляр анимации хранится в отдельном XML-файле внутри каталога res/anim.

ПРИ БОЛЬШОМ ЖЕЛАНИИ МОЖНО ДОБАВИТЬ 3 СТРАНИЦЫ КОДА АНИМАЦИИ из конца 10-11 лекции, но зачем? тут и так слишком много

23.Иерархия классов в Android SDK.

На следующей диаграмме представлена иерархия основных классов из Android SDK, с которыми предстоит иметь дело разработчику:



На самом деле классов намного больше, но это основные. Выделенные жёлтым — те, с которыми разработчик работает непосредственно (в частности, наследуются от них). Остальные так же важны, но они реже используются напрямую.
Context – это базовый абстрактный класс, реализация которого обеспечивается системой Android. Этот класс имеет методы для доступа к специфичным для конкретного приложения ресурсам и классам и служит для выполнения операций на уровне приложения, таких, как запуск активностей, отправка широковещательных сообщений, получение намерений и прочее. От класса Context наследуются такие крупные и важные

классы, как Application, Activity и Service, поэтому все его методы доступны из этих классов.

Класс View является основным строительным блоком для компонентов пользовательского интерфейса (UI), он определяет прямоугольную область экрана и отвечает за прорисовку и обработку событий. Является базовым классом для виджетов (GUI widgets), которые используются для создания интерактивных компонентов пользовательского интерфейса: кнопок, текстовых полей и т. д. А также является базовым классом для класса ViewGroup, который является невидимым контейнером для других контейнеров и виджетов, определяет свойства расположения компонентов пользовательского интерфейса. Интерфейс Android-приложения представляет собой иерархию UI компонентов, можно описать эту иерархию программно, но более простым и эффективным способом задать расположение элементов интерфейса является XML файл, который предоставляет удобную для восприятия структуру компоновки (layout file). Во время исполнения XML файл автоматически превращается в дерево соответствующих объектов.

Activity и его подклассы содержат логику, лежащую за пользовательским интерфейсом. При ближайшем рассмотрении этот класс соответствует ViewModel в архитектурном шаблоне Model-View-ViewModel (MVVM). Отношение между подклассом Activity и пользовательским интерфейсом — это отношение один к одному; обычно каждый подкласс Activity имеет только один связанный с ним слой пользовательского интерфейса, и наоборот. Код активити выполняется только когда соответствующий интерфейс пользователя видим и имеет фокус. Нет гарантии, что объект Activity и связанные с ним объекты находятся в памяти, когда активити приостановлено или завершено.

Объекты-экземпляры класса Intent используются для передачи сообщений между основными компонентами приложений. Известно, что три из четырех основных компонентов: активности, сервисы и приемники широковещательных сообщений, могут быть активированы с помощью сообщений, которые называются намерениями. Такие сообщения являются инструментом позднего связывания компонентов одного или нескольких приложений.

Экземпляр **класса Intent** представляет собой структуру данных, содержащую описание операции, которая должна быть выполнена, и обычно используется для запуска активности или сервиса. В случае с приемниками широковещательных сообщений объект Intent содержит описание события, которое произошло или было объявлено.

Класс ContentProvider и его подклассы управляют доступом к хранилищу данных. Он обеспечивает интерфейс между контент-провайдером и другими приложениями. Основное назначение этого компонента приложения заключается в предоставлении другим приложениям доступа к данным, однако ничто не мешает в приложении иметь активность, которая позволит пользователю запрашивать и изменять данные, находящиеся под управлением контент-провайдера.

Service является компонентом приложения, предназначенным для выполнения длительных операций в фоновом режиме. Они не имеют пользовательского интерфейса и

нужны в тех случаях, когда не требуется вмешательства пользователя. Сервисы работают в фоновом режиме, выполняя сетевые запросы к веб-серверу, обрабатывая информацию, запуская уведомления и т.д. Служба может быть запущена и будет продолжать работать до тех пор, пока кто-нибудь не остановит её или пока она не остановит себя сама. Сервисы предназначены для длительного существования, в отличие от активностей. Они могут работать, постоянно перезапускаясь, выполняя постоянные задачи или выполняя задачи, требующие много времени.
Класс BroadcastReceiver и его подклассы представляют собой «подписчика» в механизме взаимодейтсвия издатель/подписчик, реализованном в архитектуре Android.Этот класс рассчитан на получение объектов-намерений отправленных методом sendBroadcast()

24.Задачи и стек переходов назад. (памагитенепанимаю)

Activity (операция, действие) — это компонент приложения, определяющий экран с информацией и все действия, которые при этом может выполнить пользователь, т.е. отдельный экран, который имеет свою отдельную логику и UI.
Задача (Task) — это последовательность Activity, с которыми взаимодействует пользователь при выполнении определенного задания.
Задача — это связанный блок, который может переходить в фоновый режим, когда пользователи начинают новую задачу или переходят на главный экран с помощью кнопки Домой.

Итоги по операциям и задачам

- Когда Операция A запускает Операцию B, Операция A останавливается, но система сохраняет ее состояние. Если пользователь нажимает кнопку Назад в операции B, операция A возобновляет работу из сохраненного состояния.
- Когда пользователь выходит из задачи нажатием кнопки Домой, текущая операция останавливается и ее задача переводится в фоновый режим. Система сохраняет состояние каждой операции в задаче. Если пользователь впоследствии возобновляет задачу, выбирая значок запуска задачи, она переводится на передний план и возобновляет операцию на вершине стека.
- Если пользователь нажимает кнопку Назад, текущая операция удаляется из стека и уничтожается. Возобновляется предыдущая операция в стеке. Когда операция уничтожается, система не сохраняет состояние операции.
- Можно создавать несколько экземпляров операции, даже из других задач.

Определение режимов запуска
Использование файла манифеста:

taskAffinity, launchMode, allowTaskReparenting, , alwaysRetainTaskState, finishOnTaskLaunch, clearTaskOnLaunch

Использование флагов намерений:
FLAG_ACTIVITY_NEW_TASK, FLAG_ACTIVITY_CLEAR_TOP,
FLAG_ACTIVITY_SINGLE_TOP

Атрибут launchMode

- "standard" - поведение по умолчанию. Система всегда создаёт новую Activity и добавляет её в верх стека.
- "singleTop" - при повторном вызове Activity, которая находятся в вершине стека она не будет создаваться заново.
- "singleTask" - таск состоит из одной Activity
- "singleInstance" - Activity может быть в таске в единственном экземпляре

clearTaskOnLaunch(для корневой активити) - будет обязывать систему уничтожать все не корневые Activity у task'а при повторном запуске приложения.
finishOnTaskLaunch - будет уничтожена эта Activity

android:allowTaskReparenting и android:taskAffinity

Параметр allowTaskReparenting разрешает привязать вызванную из task'а №1 Activity до этого созданную в task'е №2 (т.е. привязанную к нему) к task'у №1.

alwaysRetainTaskState - таск не будет уничтожен системой из-за неактивности

25.Фрагменты и управление фрагментами.

Почему рекомендуется использовать Fragment?

- Помогают создавать адаптивный пользовательский интерфейс;
- Являются контроллерами представлений, содержат куски бизнес-логики, которые могут быть протестированы;
- API фрагментов предоставляет возможности работы с backstack, аналогичные Activity;
- Построены на обычных представлениях (views);
- Google рекомендовал фрагменты к использованию.

Добавление фрагмента в Activity

● объявив фрагмент в файле макета операции
●или программным образом, добавив фрагмент в существующий объект ViewGroup:
Добавление фрагмента, не имеющего пользовательского интерфейса
●Добавить фрагмент - метод add(Fragment, String);

●Получить фрагмент - метод метод findFragmentByTag().

Управление фрагментами

Класс FragmentManager

●получать фрагменты, имеющиеся в операции, с помощью метода findFragmentById() или findFragmentByTag();

●снимать фрагменты со стека переходов назад методом popBackStack();

●регистрировать процесс-слушатель изменений в стеке переходов назад при помощи метода addOnBackStackChangedListener().

Выполнение транзакций с фрагментами

```
FragmentManager fragmentManager = getFragmentManager();

FragmentTransaction fragmentTransaction =
fragmentManager.beginTransaction();

// Create new fragment and transaction
Fragment newFragment = new ExampleFragment();
FragmentTransaction transaction =
getFragmentManager().beginTransaction();
// Replace whatever is in the fragment_container view with this
fragment,
// and add the transaction to the back stack
transaction.replace(R.id.fragment_container, newFragment);
transaction.addToBackStack(null);
// Commit the transaction
transaction.commit();
```

Порядок добавления изменений к объекту fragmentTransaction:
●метод commit() должен быть вызван в последнюю очередь;
●если в один контейнер добавляется несколько фрагментов, то порядок их добавления определяет порядок, в котором они появляются в иерархии видов.

26.Намерения в Android-приложении: класс Intent и методы класса.

Intent представляет собой объект обмена сообщениями, с помощью которого можно запросить выполнение действия у компонента другого приложения.

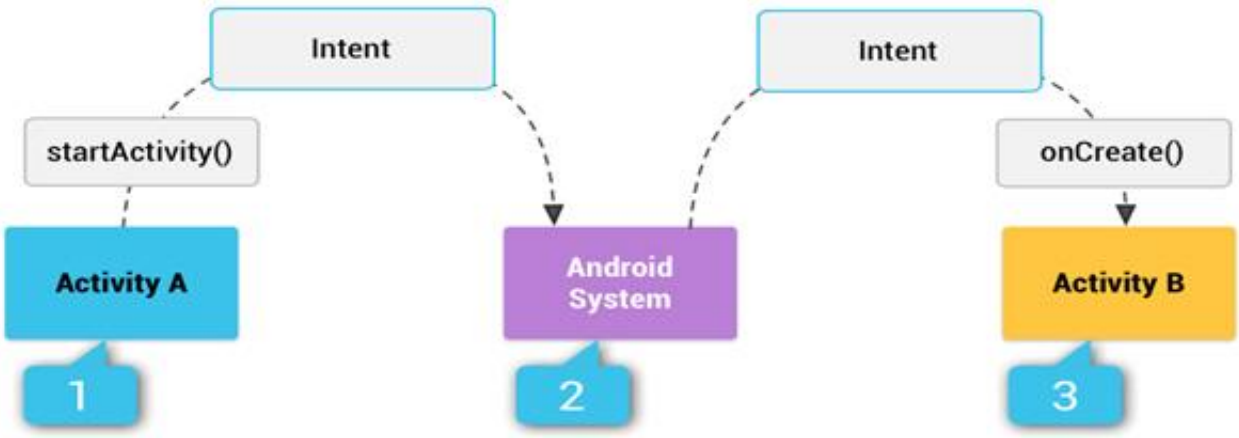
Класс Intent(Намерение)

- Используются для передачи сообщений между основными компонентами приложений.
- Содержит описание операции, которая должна быть выполнена, и обычно используется для запуска активности или сервиса.

Механизмы передачи намерений:

- Запуск активности, новое действие Context.startActivity() или Activity.startActivityForResult()
- Запуск сервиса Context.startService(), связь вызывающего компонента и сервиса Context.bindService()
- Доставка объекта-намерения приемникам широковещательных сообщений Context.sendOrderedBroadcast(), Context.sendStickyBroadcast(), Context.sendBroadcast()

В этой системе сообщений не случается накладок: сообщение-намерение, отправленное определенному компоненту, будет получено именно этим компонентом и никем другим.



27.Активности в Android-приложении: класс Activity и методы класса.

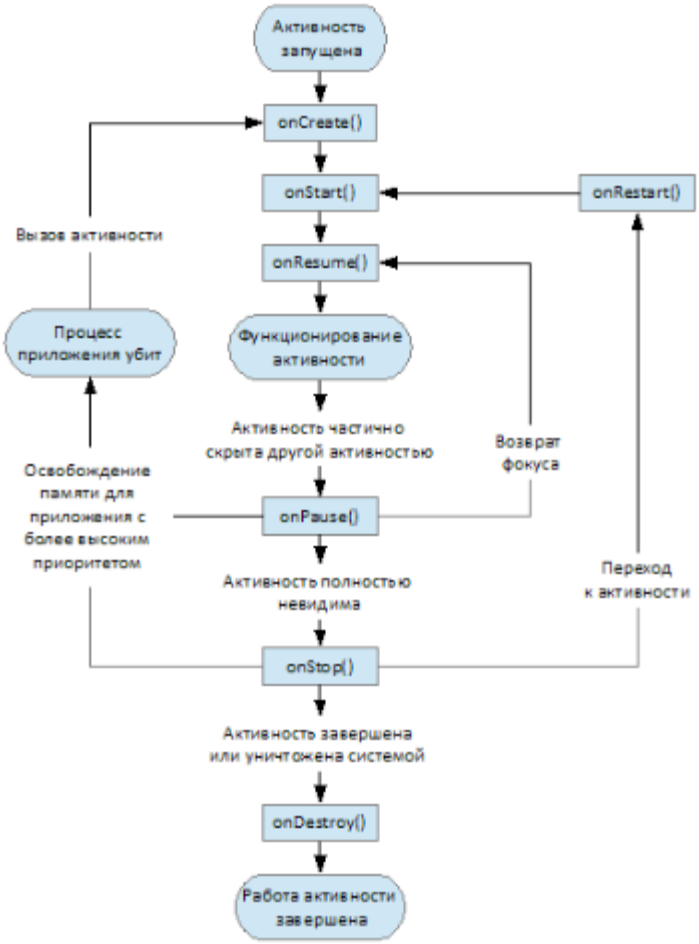
Активности (Activities)

- Активность это окно, несущее графический интерфейс пользователя.
- Обычно занимает весь экран устройства, однако возможно создавать полупрозрачные или плавающие диалоговые окна.
- Мобильные приложения обычно содержат несколько активностей.
- Одна из активностей определяется как «главная», и именно ее пользователь видит при первом запуске приложения.

Методы Activies:

- onCreate() - вызывается при создании активности. Необходимо инициализировать setContentView()
- onRestart()
- onStart()
- onResume()
- onStop()
- onDestroy()
- onPause()

Жизненный цикл активности



28.Сервисы в Android-приложении: класс Service и методы класса.

Сервис - компонент, который работает в фоновом режиме, выполняет длительные по времени операции или работу для удаленных процессов. Компонент, предназначенный для выполнения длительных операций в фоновом режиме.

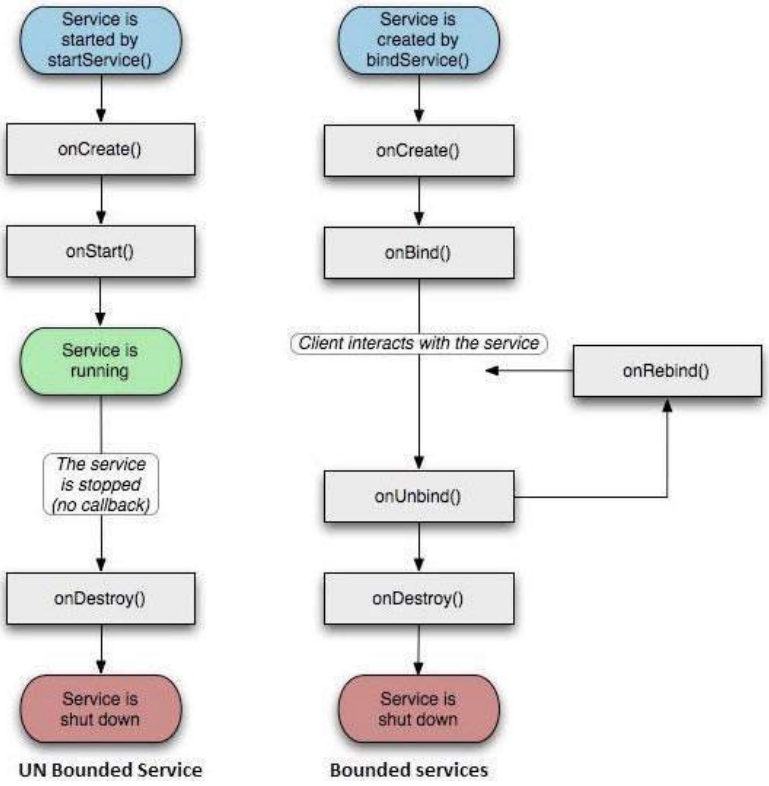
Способы существования сервисов:

● сервис запущен (started) и работает в фоновом режиме пока не выполнит свою задачу;

● сервис привязан (bound) к одному/нескольким компонентам, предлагает интерфейс для взаимодействия с компонентом и работает пока привязан хотя бы к одному компоненту.

Пример: Проигрывание музыки в фоновом режиме. Сервис может быть запущен другим компонентом и после этого работать самостоятельно, а может остаться связанным с этим компонентом и взаимодействовать с ним.

Жизненный цикл сервисов:



Типы сервисов:
● Foreground Service — сервис, выполняющий действие, состояние которого важно для пользователя.
● Background Service — сервис, выполняющий фоновое действие, состояние которого не интересует пользователя.
● Bound Service — сервис, обрабатывающий входящее Binder-подключение.

Методы класса Service:

- onStartCommand()
- onBind()
- onCreate()
- onDestroy()

29.Контент-провайдеры. Класс ContentProvider и методы класса.

- Контент - провайдер - компонент , позволяющий приложению предоставлять другим приложениям доступ к данным, которыми оно управляет.
- Управляет распределенным множеством данных приложения.
- Данные могут храниться в файловой системе , базе данных , в сети.
- Позволяет другим приложениям при наличии права доступа делать запросы или даже менять данные.

- Управляет доступом к хранилищу данных
- Класс ContentProvider - обеспечивает интерфейс между контент-провайдером и другими приложениями

Необходимы в следующих случаях:

1. приложение предоставляет сложные данные или файлы другим приложениям
2. приложение позволяет пользователям копировать сложные данные в другие приложения
3. приложение предоставляет специальные варианты поиска, используя поисковую платформу (framework)

Проектирование способа хранения данных:

- Если данные представлены файлом, то провайдер может возвращать ссылку на файл.
- Если данные представлены некоторой структурой, необходимо хранить данные в табличной форме.

Создание класса-наследника от класса ContentProvider :

- query() - извлекает данные из провайдера
- insert() - добавляет новую строку

- update() - обновляет строки
- delete() – удаляет строки
- getType() - возвращает String в формате MIME
- onCreate() - инициализацию провайдера

30.Приёмники широковещательных сообщений (Broadcast Receivers) и методы класса.

Broadcast receiver (приемник широковещательных сообщений) — компонент, позволяющий приложению принимать широковещательные извещения (broadcast'ы), специальный вид сообщений от системы или других приложений.

Исходно broadcast'ы, как следует из названия, в основном использовались для рассылки широковещательных сообщений всем подписавшимся приложениям — например, система посылает сообщение `AIRPLANE_MODE_CHANGED` при включении или отключении самолётного режима.

Сейчас вместо подписки на такие broadcast'ы, как `NEW_PICTURE` и `NEW_VIDEO`, приложения должны использовать JobScheduler. Broadcast'ы используются либо для более редких событий (таких как `BOOT_COMPLETED`), либо с явными intent'ами, то есть именно в качестве сообщения от одного приложения к другому.

Приемник — компонент, который реагирует на широковещательные извещения.

Например: Извещение о низком заряде батареи

Характеристики приемников:

- Инициирование широковещания
- Не отображают пользовательского интерфейса

- Могут создавать уведомление на панели состояний

Методы:

Есть два основных способа отсылки сообщений:

- **Context.sendBroadcast - полностью асинхронно. Все слушатели сообщения принимают его (сообщение) в неопределенном порядке, часто в одно и то же время.**
- **Context.sendOrderedBroadcast сообщения посылаются одному слушателю в конкретный момент времени. Каждый слушатель может изменить сообщение, которое получат следующие слушатели или полностью прервать трансляцию.**

метод sendBroadcast() -передав ему в качестве параметра созданный объект Intent.

31. Основы разработки интерфейсов мобильных приложений.

Визуальный дизайн интерфейса:

Строительные блоки визуального дизайна:

1. Форма
2. Размер
3. Цвет
4. Яркость
5. Направление
6. Текстура
7. Расположение

Элементы управления и дизайн навигации:

● Элементы управления – это доступные для манипулирования самодостаточные экранные объекты, посредством которых люди взаимодействуют с цифровыми продуктами.

● Controls/widgets (сокращение от windows gadgets – оконные приспособления) – это базовые строительные блоки графического пользовательского интерфейса.

Командные элементы управления

● Немедленно выполняют действие
● Часто особым образом выделяются кнопки по умолчанию
● Рекомендуется изменять внешний вид нажатой кнопки

Примеры: кнопки-значки, текстовые гиперссылки.

Элементы управления выбором:

● Флажки
● Выключатели
● Триггеры
● Радиокнопки
● Списки и раскрывающиеся списки
● Комбо-элементы

Элементы ввода:
●Ограничивающие элементы ввода
    ○Счетчики
    ○Рукоятки и ползунки
●Неограничивающие элементы ввода

Элементы управления отображением:
● Текстовые элементы
● Полосы прокрутки
● Разделители
● Выдвижные панели

Размеры элементов управления:
● Минимальный размер элемента управления 48dp
● На реальном устройстве это 7-10 мм
● Расстояние между элементами управления кратно 8dp

Материальный дизайн (Material Design) - это совершенно иной подход, обеспечивающий унифицированный набор правил, начиная от вида программного обеспечения и заканчивая функциями.

4 основных принципа:
- Тактильные поверхности
- Полиграфический дизайн
- Анимация
- Адаптивная верстка

32.Рекомендации по проектированию GUI-интерфейсов для Android-приложений.

- Реальные объекты лучше, чем кнопки и меню
- Картинки работают быстрее, чем слова

- Используйте короткие фразы, состоящие из простых слов
- Никогда не теряйте пользовательскую информацию! Если человеку придется вводить данные повторно, велика вероятность того, что он откажется использовать ваше приложение.
- Если объекты похожи, они должны выполнять сходные действия
- Показывайте только то, что необходимо пользователю именно в этот момент
- Выводите пользователю сообщения, только если вопрос действительно важен
- Делайте важные вещи быстро
- Разбивайте сложные задачи простые
- Будьте вежливы и корректны в общении с пользователем
- Пользователь всегда должен быть уверен в том, что он знает, где сейчас находится.
- Используйте интерфейсные элементы, которые будут работать в любой ситуации
- Самый главный принцип – НЕ УСЛОЖНЯЙТЕ пользователю жизнь!

Материальный дизайн - это совершенно иной подход, обеспечивающий унифицированный набор правил, начиная от вида программного обеспечения и заканчивая функциями.

4 основных принципа:

- Тактильные поверхности
- Полиграфический дизайн
- Анимация
- Адаптивная верстка

33.Основы разработки многооконных приложений.

Часто невозможно разместить все элементы так, чтобы их можно было увидеть одновременно.
Для разделения интерфейса на части используют следующие подходы:

- Диалоговые окна
- Несколько активностей
- Swipe

Диалоговые окна:
Взаимодействуем с пользователем при помощи сообщений.
Диалоговое окно позволяет пользователю ввести или получить информацию.

Работает в модальном режиме.
Виды диалоговых окон:

- Класс Dialog и его производные:
  - AlertDialog
  - DatePickerDialog или TimePickerDialog
  - ProgressDialog
- Уведомления (notifications)
- Всплывающие подсказки (toasts)

Несколько активностей:

- Необходимо тщательное проектирование распределения функционала между активностями
- Не стоит перегружать один экран информацией
- Если на активности планируется только одно поле для ввода, ее стоит заменить диалоговым окном

Переключение между активностями:

- При помощи кнопок или других элементов управления
- С использованием сенсорного экрана

Работа с несколькими активностями:

- Для вызова другой активности необходимо вручную править файл манифеста
- Для каждой новой активности необходимо занести информацию о ее имени и названии xml-файла, в котором она описана
- При загрузке приложения первой появляется активность, чье описание находится первым в манифесте!

Swipe:

- Несколько экранов отображаются по очереди, используя одну и ту же активность
- Не нужно править файл манифеста
- Для каждого экрана необходимо сделать свой xml-файл с описанием
- Перемещение между частями активности возможно как с помощью кнопок, так и перелистыванием

34.Разметка (макеты) страниц и основные элементы GUI.

Activity состоит
- Views
- ViewGroups
- Разметки наследуются от класса android.view.ViewGroup
- Класс android.view.ViewGroup наследуется от android.view.View

Представления View
- TextView
- EditText
- ImageView
- Управляющие элементы:
  - Простые:
    - Button
    - CheckBox
    - RadioButton
    - Toggle button
    - Pickers
  - Списки:
    - ListView
    - Spinner
    - ViewFlipper
    - GridView
    - Gallery

ViewGroups — разметка
- FrameLayout - разметка для отображения одного элемента.
- LinearLayout - разметка для отображения одного или нескольких элементов в одну линию, горизонтально или вертикально.
- TableLayout - разметка для расположения элементов в виде таблицы.
- RelativeLayout - разметка для расположения элементов относительно родителя или друг друга.
- GridLayout - имеет колонки, ряды, клетки как в TableLayout, но при этом элементы могут гибко настраиваться.
- ScrollView - разметка для расположения элементов относительно родителя или друг друга.

Свойства разметки:
layout_width
layout_height
layout_marginTop

layout_marginBottom
layout_marginLeft
layout_marginRight
layout_gravity
layout_weight
layout_x
layout_y

Расположение относительно родительского элемента (RelativeLayout)
●android:layout_alignParentTop
●android:layout_alignParentLeft
●android:layout_alignParentRight
●android:layout_alignParentBottom
●android:layout_centerInParent
●android:layout_centerHorizontal
●android:layout_centerVertical
Расположение относительно других элементов (RelativeLayout)
●android:layout_above
●android:layout_toLeftOf
●android:layout_toRightOf
●android:layout_below
●android:layout_alignBaseline
●android:layout_alignTop
●android:layout_alignLeft
●android:layout_alignRight
●android:layout_alignBottom

Android:layout_width, android:layout_height
Значения атрибута:
●fill_parent
●wrap_content
●точное значение размера

Единицы измерения:
●dp - (Density-independent Pixels)
●sp - (Scale-independent Pixels)
●px — пиксели
●mm - миллиметры
●in - дюймы
●pt — (points) 1/72 дюйма

android:layout_gravity - атрибут, определяющий как данный View позиционируется внутри родительского элемента (top, bottom, left, right, center)

margin - отступ, **padding** - отступ внутри элемента

android:weight - атрибут определяет соотношение, в котором будет распределятся пространство для элементов разметки.

Тип элемента управления	Описание	Связанные классы
Кнопка	Кнопка, которую пользователь может нажать для выполнения действия.	`Button`
Текстовое поле	Редактируемое текстовое поле. Можно воспользоваться виджетом `AutoCompleteTextView`, чтобы создать виджет для ввода текста с возможностью автозаполнения с помощью подсказок.	`EditText`, `AutoCompleteTextView`
Флажок	Переключатель, которым можно воспользоваться для включения или отключения функции или компонента. Флажки следует использовать при отображении группы доступных для выбора параметров, которые не являются взаимоисключающими.	`CheckBox`
Переключатель	Этот элемент управления аналогичен флажку, за исключением того, что в группе элементов можно выбрать только один вариант.	`RadioGroup` `RadioButton`
Кнопка-переключатель	Кнопка включения/отключения с индикатором.	`ToggleButton`
Раскрывающийся список	Раскрывающийся список параметров, в котором пользователь может выбрать только одно значение.	`Spinner`
Элементы выбора	Элементы диалогового окна для выбора одного значения из набора с помощью кнопок со стрелками вверх и вниз или с помощью жеста пролистывания. Для ввода значений даты (дня, месяца, года) используйте виджет `DatePicker`, а для ввода значений времени (часов, минут и указания времени до полудня или после [AM/PM]) — виджет `TimePicker`. Формат этих виджетов выбирается автоматически на основе региональных настроек на устройстве.	`DatePicker`, `TimePicker`

35.Сенсорное управление в смартфонах.

Одной из отличительных особенностей позволяющих назвать мобильный телефон смартфоном - это (сенсорный) touch-интерфейс

Touch-интерфейс

- интерфейс, основанный на виртуальных элементах управления
- выбор выполняется
  - простым касанием
  - жестами(gestures)
- если точек касания несколько (т. е. используется несколько пальцев), интерфейс называется **multi-touch**

Сенсорное управление

Подразумевает использование сенсорных жестов для взаимодействия с приложением.

Виды жестов:

1. **Касание** (touch)
   Использование: Запуск действия по умолчанию для выбранного элемента

Выполнение: нажать, отпустить

2. **Длинное касание** (long touch)
   Использование: Выбор элемента. Не стоит использовать этот жест для вызова контекстного меню
   Выполнение: нажать, ждать, отпустить
3. **Скольжение** или **перетаскивание** (swipe or drag)
   Использование: Прокрутка содержимого или навигация между элементами интерфейса одного уровня иерархии
   Выполнение: нажать, переместить, отпустить
4. **Скольжение после длинного касания** (long press drag)
   Использование: Перегруппировка данных или перемещение в контейнер
   Выполнение: длительное касание, переместить, отпустить
5. **Двойное касание** (double touch)
   Использование: Увеличение масштаба, выделение текста
   Выполнение: быстрая последовательность двух касаний
6. **Перетаскивание с двойным касанием** (double touch drag)
   Использование: Изменение размеров: расширение или сужение по отношению к центру жеста
   Выполнение: касание, следующее за двойным касанием со смещением:
   a. смещение вверх уменьшает размер содержимого
   b. смещение вниз увеличивает размер содержимого
7. **Сведение пальцев** (pinch close)
   Использование: уменьшение содержимого, сворачивание
   Выполнение: касание экрана двумя пальцами, свести, отпустить
8. **Разведение пальцев** (pinch open)
   Использование: увеличение содержимого, разворачивание.
   Выполнение: касание экрана двумя пальцами, развести, отпустить

Процесс распознавания жестов

- сбор данных
- распознавание жеста

Основные действия пользователя при взаимодействии с сенсорным экраном:

- касание экрана пальцем
- перемещение пальца по экрану
- отпускание пальца
    - Эти действия - сенсорные события (touch-события)

Сенсорные события (touch-события)

Жест

- **начинается** при первом касании экрана
- **продолжается** пока система отслеживает положение пальцев пользователя
- **заканчивается** получением финального события, состоящего в том, что ни один палец не касается экрана
- **инициируют** вызов метода onTouchEvent()
- **обрабатываются**, если этот метод реализован в классе активности или некоторого компонента, иначе событие просто игнорируется
- Объект **MotionEvent**, передаваемый
- в метод **onTouchEvent()**, предоставляет детали каждого взаимодействия
- Константы класса MotionEvent, определяющие сенсорные события:
    - **MotionEvent.ACTION_DOWN** — касание экрана пальцем, начальная точка для любого сенсорного события или жеста
    - **MotionEvent.ACTION_MOVE** — перемещение пальца по экрану
    - **MotionEvent.ACTION_UP** — поднятие пальца от экрана

Процесс распознавания жеста. Обработка событий для распознавания жеста

- своя собственная обработка событий — можно работать с произвольными жестами
- стандартные жесты без обработки — отдельных сенсорных событий класс GestureDetector

Класс GestureDetector для распознавания стандартных жестов:

- поддерживает жесты: onDown(), onLongPress(), onFling() и т. д.
- может использоваться в связке с методом onTouchEvent()

API для работы с жестами

- предоставляется Android, начиная с версии 1.6
- располагается в пакете android.gesture
- позволяет сохранять, загружать, создавать и распознавать жесты

Предустановленное приложение Gesture Builder по созданию жестов

- содержится в Виртуальном устройстве Android (AVD)
- созданные жесты сохраняются на SD карте виртуального устройства
- созданные жесты могут быть добавлены в приложение в виде бинарного ресурса

Виртуальное устройство Android (AVD)

- Для распознавания жестов необходимо добавить компонент GestureOverlayView в XML файл активности:

- как обычный элемент графического интерфейса пользователя (встроен в компоновку, например RelativeLayout)
- как прозрачный слой поверх других компонентов (как корневой элемент в XML файле активности)
- При использовании собственных жестов в приложении необходимо реализовать интерфейс OnGesturePerformedListener и его метод onGesturePerformed()

36.Хранение данных в ОС Android на примере Shared preferences.

Shared preferences

· Файл, содержащий пары "ключ-значение", и предоставляет простые методы для чтения и записи;
· Управление каждым файлом SharedPreferences осуществляется с помощью инфраструктуры и может быть частным или общим.

Вы можете создать новый файл общих настроек или получить доступ к существующему, вызвав один из следующих методов:

· getSharedPreferences () - Используйте это, если вам нужно несколько общих файлов настроек, идентифицируемых по имени, которое вы указываете в первом параметре. Вы можете вызвать это из любого контекста в вашем приложении.

Параметры

1. name — выбранный файл настроек
2. mode — режим работы:

MODE_PRIVATE
MODE_WORLD_READABLE
MODE_WORLD_WRITEABLE
MODE_MULTI_PROCESS

· getPreferences () - Используйте это из Activity, если вам нужно использовать только один общий файл предпочтений для Activity. Поскольку при этом извлекается файл общих настроек по умолчанию, который принадлежит Activity, вам не нужно указывать имя.

Все эти методы возвращают экземпляр класса **SharedPreferences**, из которого можно получить соответствующую настройку с помощью ряда методов:

- getBoolean(String key, boolean defValue);
- getFloat(String key, float defValue);
- getInt(String key, int defValue);
- getLong(String key, long defValue);
- getString(String key, String defValue)

Для изменения файла нужно создать Editor вызвав edit() на объекте SharedPreferences.

Далее с помощью этих методов добавляем новую информацию:
· putBoolean(String key, boolean value),
· putFloat(String key, float value),
· putInt(String key, int value),
· putLong(String key, long value),
· putString(String key, String value),
· putStringSet(String key, Set values)

Для сохранения изменений нужно добавить editor.commit().

Для очистки значений используйте методы **SharedPreferences.Editor.remove(String key)** и **SharedPreferences.Editor.clear()**.

Строка позволяющая получить контейнер со всеми записями:
Map<String, ?> allPreferences = mySharedPreferences.getAll();

37.Хранение данных в ОС Android на примере внутреннего и внешнего хранилища.

- Почти каждому приложению необходимо место для хранения данных.
- Каждое приложение на устройстве Android имеет каталог в своей песочнице (sandbox). Хранение файлов в песочнице защищает их от других приложений
- Песочница каждого приложения представляет собой подкаталог каталога /data/data, имя которого соответствует имени пакета приложения. Например, для приложения TestIntent полный путь к каталогу песочницы имеет вид /data/data/com.bignerdranch.android.testintent
- Кроме песочницы, приложение может хранить файлы во внешнем хранилище, например SD-карте.

- Доступ к файлам на внешнем хранилище имеет любой желающий.

Хранение и загрузка файлов

Внутренняя память

Смысл следует непосредственно из названия. Внутреннее хранилище (internal storage) располагается всегда в памяти смартфона вне зависимости от того, есть ли возможность установки карты памяти (и тем более того, вставлена ли она). Эта область памяти является защищенной. Находится в системном разделе /data. По умолчанию все файлы, которые там располагаются, доступны только тому приложению, которое их создало. Разумеется, можно сделать файлы доступными для других приложений, но это надо делать специально. Если приложение не открывает файлы для доступа извне, достучаться к ним можно будет только получив root.
Назначение хранилища понятно: внутренние защищенные данные, к которым не должно быть нерегламентированного доступа.

- Всегда доступна.
- Сохраненные здесь файлы по умолчанию доступны только вашему приложению.
- При удалении пользователем вашего приложения система Android удаляет из внутренней памяти все файлы этого приложения.

Расположение
/data/data/%package%/

Например, /data/data/com.bignerdranch.android.testintent

- Структура
  - lib
  - cache
  - databases
- Безопасность
  - Context.MODE_PRIVATE
  - Context.MODE_WORLD_READABLE
  - Context.MODE_WORLD_WRITEABLE
- getFilesDir()
- getCacheDir()

Создание файла с помощью конструктора File:

```
File file = new File(context.getFilesDir(), filename);
```

Внешняя память

внешнее хранилище необязательно размещается на карте памяти. Это может быть и внутренняя память смартфона, но весь раздел с такими данными размещается в общем доступе.

Плюсы такого подхода очевидны: данные доступны извне для целей пользователя. А если это карта памяти, то и емкость может быть ограничена только вашими финансами .

Минусы тоже понятны: в любой момент любое приложение (конечно, имеющее разрешение на доступ к «внешним» данным) может взять и стереть чужие файлы. Также файлы будут удалены системой при удалении приложения (или при очистке его данных).

- Доступно не всегда.
- Такие хранилища доступны для чтения везде.
- При удалении пользователем вашего приложения система Android удаляет из внешних хранилищ файлы этого приложения, только если они сохраняются в директории из getExternalFilesDir().

- Папка cache
- Уровни доступа
  - Приложение (/sdcard/Android/data/%package%)
  - Общие (Environment.DIRECTORY_*)
- Изменения в Android KitKat 4.4 (API 19)
  - READ_EXTERNAL_STORAGE
  - WRITE_EXTERNAL_STORAGE

Использование внешнего хранилища

- Внешнее хранилище удобно для передачи данных (музыки, фотографий, загруженной из Интернета информации) другим приложениям или пользователям.
- Для записи во внешнее хранилище необходимо:
  - убедиться в том, что внешнее хранилище доступно, например, используя класс android.os.Environment;
  - получить дескриптор каталога внешних файлов (метод класса Context,

который предоставит вам эту информацию).

38.Основы работы с базами данных SQLite. Запросы в sqlite.

Небольшая и при этом мощная система управления базами данных

Плюсы:

- Не требует установки
- Не требует администрирования
- Бесплатная
- Маленькая

Пакеты для работы с SqLite:

- android.database.sqlite — классы SQLiteCursor, SQLiteDatabase, SQLiteQuery, SQLiteQueryBuilder и SQLiteStatement;
- android.database.

Android хранит файл базы данных в папке: data/data/packagename/databases/

Типы данных в SqLite:

- NULL
- INTEGER
- REAL
- TEXT
- NUMERIC
- BLOB

Способы доступа к базе данных sqlite в Android Studio

- Сохранить локально и открыть в SQLite Browser
- Доступ из консоли adb

- Плагин SQLScout от JetBrains
- Плагин для отладки Android-приложений Stetho

Виды sql запросов:
1. DDL
2. Modification
3. Query

DDL

- Такие запросы используются для создания таблиц
- В файле базы данных может быть несколько таблиц

Создание таблицы:

```
create Table_Name (
_id  integer primary key autoincrement,
field_name_1 text,
field_name_2 text);
```

Modification

- Такие запросы используются для добавления, изменения или удаления записей

Добавление строки:

```
insert into Table_Name values(null, value1, value2);
```

Удаление строки:

```
delete from Table_Name;
```

Query

- Такие запросы позволяют получать выборки из таблицы по различным критериям

Пример запроса :

```
select * from Table_Name where (_id = smth);
select Field_Name_1, Field_Name_2 from Table_Name
```

Field_Name_1 = smth);

39.Классы и основные методы для работы с SQLite.

Работа с базой данных сводится к следующим задачам:

- Создание и открытие базы данных
- Создание таблицы
- Создание интерфейса для вставки данных (insert)
- Создание интерфейса для выполнения запросов (выборка данных)
- Закрытие базы данных

Класс ContentValues

Класс ContentValues используется для добавления новых строк в таблицу. Каждый объект этого класса представляет собой одну строку таблицы и выглядит как ассоциативный массив с именами столбцов и значениями, которые им соответствуют

Курсоры

В Android запросы к базе данных возвращают объекты класса Cursor. Вместо того чтобы извлекать данные и возвращать копию значений, курсоры ссылаются на результирующий набор исходных данных. Курсоры позволяют управлять текущей позицией (строкой) в результирующем наборе данных, возвращаемом при запросе.Объект Cursor, возвращаемый методом query(), обеспечивает доступ к набору записей результирующей выборки. Для обработки возвращаемых данных объект Cursor имеет набор методов для чтения каждого типа данных — getString(), getInt() и getFloat().

Класс SQLiteOpenHelper

Библиотека Android содержит абстрактный класс SQLiteOpenHelper, с помощью которого можно создавать, открывать и обновлять базы данных. Это основной класс, с которым вам придётся работать в своих проектах. При реализации этого вспомогательного класса от вас скрывается логика, на основе которой принимается решение о создании или обновлении базы данных перед ее открытием. Класс SQLiteOpenHelper содержит два обязательных абстрактных метода:

- onCreate() — вызывается при первом создании базы данных
- onUpgrade() — вызывается при модификации базы данных

Также используются другие методы класса:

- onDowngrade(SQLiteDatabase, int, int)
- onOpen(SQLiteDatabase)
- getReadableDatabase()
- getWritableDatabase()

Если база данных не существует, вспомогательный класс вызывает свой обработчик onCreate(); если версия базы данных изменилась, вызовется обработчик onUpgrade(). В любом случае вызов методов getWritableDatabase/getReadableDatabase, в зависимости от

ситуации, вернет существующую, только что созданную или обновленную базу данных.Оба метода имеют разные названия, но возвращают один и тот же объект. Только метод для чтения getReadableDatabase() сначала проверит доступность на чтение/запись. В случае ошибки метод проверит доступность на чтение и только потом уже вызовет исключение. Второй метод сразу проверяет доступ к чтению/записи и вызывает исключение при отказе в доступе.В приложении необходимо создать собственный класс, наследуемый от SQLiteOpenHelper. В этом классе необходимо реализовать указанные обязательные методы, описав в них логику создания и модификации вашей базы.

SQLiteDatabase

Для управления базой данных SQLite существует класс SQLiteDatabase. В классе SQLiteDatabaseопределены методы query(), insert(), delete() и update() для чтения, добавления, удаления, изменения данных. Кроме того, метод execSQL() позволяет выполнять любой допустимый код на языке SQL применимо к таблицам базы данных, если вы хотите провести эти (или любые другие) операции вручную.

Каждый раз, когда вы редактируете очередное значение из базы данных, нужно вызывать метод refreshQuery() для всех курсоров, которые имеют отношение к редактируемой таблице.

Для составления запроса используются два метода: rawQuery() и query(), а также

Метод rawQuery() - это сырой запрос, как есть, т.е. пишется строка запроса, как это обычно делается в SQL.

В метод query() отедельно передают семь параметров(table,columnNames,whereClause, groupBy, having,orderBy).

Метод openOrCreateDatabase: Открытие и создание баз данных без использования SQLiteOpenHelper

Вы можете открывать и создавать базы данных без помощи класса SQLiteOpenHelper, используя метод openOrCreateDatabase(), принадлежащий объекту Context вашего приложения. Получите доступ к базе данных в два шага. Сначала вызовите метод openOrCreateDatabase(), чтобы создать новую базу данных. Затем из полученного экземпляра базы данных вызовите execSQL(), чтобы выполнять команды на языке SQL, с помощью которых будут созданы таблицы и установлены отношения между ними.

40.2D и 3D графика в мобильных приложениях для ОС Android.

Холсты и графические объекты

- Изобразить графику или анимацию в элементе пользовательского интерфейса
- Изображать графику напрямую на холсте:
    - в том же потоке;
    - в отдельном потоке.

Аппаратное ускорение

Все операции рисования на холсте исполняются с использованием GPU.
Аппаратное ускорение доступно по умолчанию, если целевой уровень API больше или равен 14, но может быть включено явно.
Включение может нарушать некоторые пользовательские изображения или вызовы рисования!

OpenGL
OpenGL является кросс-платформенным API, который определяет стандартный программный интерфейс для аппаратного обеспечения, занимающегося обработкой 3D графики.
Создана для рабочих станций Unix компанией Silicon Graphics, Inc. (SGI).
1992 - первая стандартизованная версия.
Применяется во всех операционных системах и служит основой для большинства игр.
Консорциум Khronos Group управляет стандартом с 2000 года.
Разновидности стандартов: OpenGL ES и EGL Native Platform Graphics Interface

OpenGL ES API и Android
OpenGL ES 1.0 и 1.1 поддерживается Android 1.0 и выше;
OpenGL ES 2.0 поддерживается Android 2.2 (API уровень 8) и выше;
OpenGL ES 3.0 поддерживается Android 4.3 (API уровень 18) и выше.
OpenGL ES API
glVertexPointer - используется для указания серии точек (или вершин);
glDrawElements - применяется для рисования элементов при помощи примитивных контуров;
glColor - используется для указания стандартного цвета для рисунка, который предстоит создать.;
glClear - используется для очистки поверхности изображения (drawing surface);
gluLookAt - контролирует направление камеры;;
glFrustum - контролирует размер зоны видимости или степень увеличения/уменьшения;
glViewport - контролирует размер экрана или размер «пленки» в камере.
Взаимодействие OpenGL ES и Android
Классы:

- GLU.
- GLUtils.
- Matrix.
- Visibility.
- GLDebugHelper.

Интерфейсы:

- GLSurfaceView.Renderer.
- GLSurfaceView.EGLConfigChooser.
- GLSurf aceVi ew. GLWrapper.

Графические объекты в Android 5.0 (API 21)

- векторные объекты
- тонирование графических объектов
- извлечение цвета

Тонирование

Можно тонировать: растровые изображения, изображения в формате NinePatch, определенные как альфа-маски.
Можно применять: цветовые ресурсы, атрибуты темы.
Можно применить к объектам *BitmapDrawable* и *NinePatchDrawable* с помощью метода setTint().
Можно задать цвет и способ тонирования в макетах, используя для этого атрибуты *android:tint* и *android:tintMode*.

Извлечение главных цветов на изображении

Используется класс *Palette*
Цвета извлекаются в соответствии с профилем:

- насыщенные цвета;
- насыщенные темные цвета;
- насыщенные светлые цвета;
- приглушенные цвета;
- приглушенные темные цвета;
- приглушенные светлые цвета.

Для извлечения передаем объект Bitmap в статический метод Palette.generate() в фоновом потоке.

Создание векторных объектов

```
<!-- res/drawable/heart.xml -->
<vector xmlns:android="http://schemas.android.com/apk/res/android"
    <!-- intrinsic size of the drawable -->
    android:height="256dp"
    android:width="256dp"
    <!-- size of the virtual canvas -->
    android:viewportWidth="32"
    android:viewportHeight="32">

  <!-- draw a path -->
  <path android:fillColor="#8fff"
      android:pathData="M20.5,9.5
                        c-1.955,0,-3.83,1.268,-4.5,3
                        c-0.67,-1.732,-2.547,-3,-4.5,-3
                        C8.957,9.5,7,11.432,7,14
                        c0,3.53,3.793,6.257,9,11.5
                        c5.207,-5.242,9,-7.97,9,-11.5
                        C25,11.432,23.043,9.5,20.5,9.5z" />
</vector>
```

41.Анимация для платформы Android. Типы анимации.

Ресурсы анимации

●Анимация свойств (Property Animation)
●Анимация представления (View Animation):
a.Tween Animation — основана на расчете промежуточных кадров и может использоваться для поворота, перемещения, растягивания, затемнения элементов.
i.TranslateAnimation
ii.ScaleAnimation
iii.RotateAnimation
iv.AlphaAnimation
b.Frame Animation — пошаговая анимация, т.е. последовательный вывод заранее подготовленных изображений.

Каждый экземпляр анимации хранится в отдельном XML-файле внутри каталога res/anim.

Анимация в Android 2.3

- покадровая анимация
- анимация компоновки
- анимация представления

Базовые типы анимации с построением промежуточных кадров

- Анимация масштабированием
- Анимация поворотом
- Анимация преобразованием
- Анимация альфа-канала

Анимация представления

- Матрица трансформации
- Единичная матрица

Ключевые методы класса Matrix:

- matrix.reset()
- matrix.setScale()
- matrix.setTranslate()
- matrix.setRotate()
- matrix.setSkew()

Системы анимации в Android 3.0

- Анимация свойств

- Анимация компонентов пользовательского интерфейса

Ключевые концепции нового API-интерфейса анимации

- аниматоры;
- аниматоры значений;
- аниматоры объектов;
- наборы аниматоров;
- построители аниматоров;
- слушатели анимации;
- хранители значений свойств;
- анализаторы типа;
- аниматоры свойств представления;
- переходы компоновки;
- аниматоры, определенные в XML-файлах.

Анимация свойств
- Позволяет определить анимацию для изменения любого свойства объекта

  Характеристики:

  - Продолжительность
  - Временная интерполяция
  - Количество повторов и поведение
  - Группа анимаций
  - Частота обновления кадров

Класс Animator
- Предоставляет базовую структуру для создания анимации

- На прямую не используется

Класс ValueAnimator
- Потомок класса Animator

- Обеспечивает всю основную функциональность

Класс AnimatorSet

- Потомок класса Animator
- Предоставляет механизмы группировки анимаций, таким образом, что они выполняются некоторым образом относительно друг друга

Классы-вычислители

- определяют, как вычислять значения заданных свойств

  Вычислители:

  - IntEvaluator для вычисления целочисленных значений
  - FloatEvaluator для вычисления вещественных значений
  - ArgbEvaluator для вычисления значений цвета в шестнадцатеричном представлении
  - TypeEvaluator – интерфейс, позволяющий создавать собственных вычислителей

Интерполяторы

- определяют, с помощью каких функций от времени вычисляются значения свойств, для которых задается анимация
- Интерполяторы определены в пакете android.view.animation
- Если ни один из существующих интерполяторов не подходит, можно создать собственный, реализовав интерфейс TimeInterpolator

Анимация компонентов пользовательского интерфейса

- Используется для реализации анимации преобразований над наследниками класса View

- Для расчёта нужно: начальная точка, конечная точка, размер, поворот и другие общие аспекты анимации

Основные классы анимации

- AnimationSet (элемент <set>);
- AlphaAnimation (элемент <alpha>);
- RotateAnimation (элемент <rotate>);
- ScaleAnimation (элемент <scale>);
- TranslateAnimation (элемент <translate>).

Атрибуты анимации

- duration;
- startOffset;
- fillBefore;
- fillAfter;
- repeatCount;
- repeatMode;
- zAdjustment;
- interpolator.

Анимация для Android 5.0 (уровень API 21)

- реакция на касание;
- круговое появление;
- переходы;
- перемещение по кривой;
- изменение состояний представления.

42.Использование встроенной камеры и работа с мультимедиа в Android приложениях.

- Платформа Android позволяет приложениям получать фотографии и

записывать видео

- Для решения этих задач, существует два способа:
    - непосредственное обращение к камере
    - использование намерений (Intent) для вызова существующего приложения

Классы объектов по работе с камерой

Camera — класс, реализующий управление камерами устройства (для получения фотографий или записи видео)

SurfaceView — класс, используемый для предоставления пользователю возможности предварительного просмотра

MediaRecorder – класс, используемый для записи видео с камеры

Intent – класс, содержащий абстрактное описание выполняемой операции, оно передается системе Android, а ОС сама находит и запускает необходимое приложение и возвращает результат его работы

Типы намерений по работе с камерой

- MediaStore.ACTION_IMAGE_CAPTURE – для запроса на выполнение фотоснимков
- MediaStore.ACTION_VIDEO_CAPTURE – для запроса на запись видео

Мультимедиа библиотека Android

- позволяет легко использовать в приложениях аудио, видео и изображения:
    - из медиа файлов сохраненных как ресурсы приложения (raw ресурсы)
    - из файлов, расположенных в файловой системе
    - из потока данных, получаемого через сетевое соединение
- НО! невозможно воспроизводить аудио во время звонка
- Для воспроизведения аудио и видео Android предоставляет класс MediaPlayer
- При работе с аудиоконтентом можно воспроизводить необработанные данные (проигрывание динамически генерируемого аудио)
- Для записи аудио и видео Android предоставляет класс MediaRecorder

43.Обзор сенсоров и датчиков и взаимодействие с системами позиционирования.

Взаимодействие с системами позиционирования

- **Системы позиционирования** - позволяют определить местоположение в некоторой системе координат (обычно широта и долгота)
- **Системы позиционирования смартфона**
    - смартфон постоянно связывается с сотовой вышкой, в зоне действия которой он находится
    - У каждой вышки в мире свой уникальный идентификатор - идентификатор соты (Cell ID), для нее точно известны широта и долгота ее расположения
    - зная Cell ID метоположения, можно получить географические координаты центра этой соты
    - Радиусы сот варьируются от активности сетевой трафика района
    - Результаты приближенные («плюс-минус трамвайная остановка»)
    - Если смартфон в зоне действия более, чем одной сотовой вышки, возможно выполнение триангуляции его местоположения
    - сотовая вышка может определить, с какого направления приходит сигнал – точное местоположение без установки дополнительного оборудования
    - спутниковые системы глобального позиционирования (Global Positioning System, GPS):
        - GPS, разработанная и реализованная в США
        - система ГЛОНАСС (Глобальная навигационная спутниковая система) - советско-российская спутниковая система навигации
    - Многие смартфоны поддерживают обе системы GPS – надежность и точность определения координат, прежде всего, в городских условиях
    - Есть возможность использования сигналов WiFi, Bluetooth и NFC, а также внутреннего сенсора для более точной геолокации, особенно внутри помещений

Приложения по учету текущего местоположения под Android

- спутниковые системы глобального позиционирования (GPS)
- определение местоположения в сети (с помощью Network Location Provider)

Спутниковые системы глобального позиционирования (GPS)

- GPS дает более точные результаты

НО

- плохо работает в помещениях (чаще не работает)
- сильно расходует заряд батареи
- медленно определяет координаты

Network Location Provider

- определяет координаты, используя сигналы сотовых вышек и WiFi,
- может работать как на улице, так и внутри помещений
- более экономно расходует заряд батареи
- работает быстрее по сравнению с GPS

Доступ к геолокации смартфона

- Осуществляется через классы пакета android.location
- Центральный класс пакета LocationManager - доступ к системным сервисам для определения координат устройства
- Добавление карт с помощью Google Maps Android API - автоматический доступ к серверам Google Maps

Google Maps Android API

- загрузка данных
- отображение карт
- сенсорные жесты на карте
- добавление маркеров, многоугольников и внешних прозрачных слоев
- изменение пользовательского представления отдельных участков карты
- Ключевой класс MapView
    - отображает карту с данными полученными из сервиса Google Maps
    - предоставляет все элементы пользовательского интерфейса, необходимые для управления картой
- Когда MapView в фокусе
    - Он перехватывает нажатия клавиш и сенсорные жесты для выполнения автоматического перемещения и изменения масштаба карты
    - Он управляет сетевыми запросами для получения дополнительных фрагментов карты
- не является частью платформы Android
- доступен на любом устройстве с Google Play Store, работающем, начиная с Android 2.2, через Google Play services
- для интеграции в приложения, в Android SDK необходимо установить библиотеку Google Play services

Другие сенсоры и датчики

Сенсоры могут быть полезны, если необходимо регистрировать

- положение и перемещения

- повороты устройства в трехмерном пространстве
- изменения параметров окружающей среды

3 категории сенсоров

- Датчики движения - измеряют силы ускорения и вращательные силы по трем осям
    - Это акселерометры, гироскопы, датчики вектора вращения и сенсоры силы тяжести
- Датчики окружающей среды - измеряют различные параметры окружающей среды, (температура воздуха и давление, освещенность и влажность)
    - Это барометры, термометры и датчики освещенности
- Датчики положения - измеряют физическое положение устройства
    - Это магнитометры и датчики ориентации устройства в пространстве

Реализация сенсоров и датчиков

- **Аппаратно-реализованные датчики** - физические элементы встроенные в мобильное устройство, получают данные путем прямых измерений свойств (ускорение, сила геомагнитного поля, изменение углов)
- **Программно-реализованные датчики** – получают данные с одного или нескольких физических датчиков и вычисляют значение, которое от них ожидается
- Какие типы датчиков поддерживаются Android можно узнать по ссылке: http://developer.android.com/guide/topics/sensors/sensors_overview.html

Набор классов и интерфейсов для работы с сенсорами

- является частью пакета android.hardware
- **позволяет**:
    - определять какие сенсоры доступны на устройстве
    - определять индивидуальные возможности сенсоров, такие как максимальное значение, производитель, требования к потребляемой энергии и разрешения
    - собирать данные с сенсоров и определять минимальную частоту, с которой выполняется сбор данных
    - подключать и отключать слушателей событий от датчиков, события состоят в изменении значений датчиков

Классы и интерфейсы по работе с датчиками

- **SensorManager**
    - создает экземпляр сервиса, связанного с сенсором
    - предоставляет различные методы для доступа и составления списка сенсоров, подключения и отключения слушателей событий от сенсоров, сбора информации
    - содержит константы для задания точности сенсора, частоты получения данных и настройки датчиков

- **Sensor** - для создания экземпляра датчика, предоставляет методы, позволяющие определить свойства сенсора
- **SensorEvent** - для создания объекта, соответствующего событию датчика и предоставляющего следующую информацию: данные сенсора; тип сенсора, породившего событие, точность данных и время появления события
- **Интерфейс SensorEventListener** - для реализации двух методов, получающих уведомления (события датчиков), когда меняется значение сенсора или когда меняется точность сенсора.

44.Использование библиотек при разработке Android-приложений.

● Библиотека — сборник подпрограмм или объектов, используемых для разработки программного обеспечения (ПО)

● Для ОС Android существует большое количество подключаемых библиотек

● Их можно классифицировать в зависимости от назначения

Виды библиотек:

● Библиотеки совместимости

● Библиотеки специального назначения

● Библиотеки, предоставляющие дополнительные возможности

Подключение библиотек:
● Библиотеки могут поставляться как в собранном и уже готовом к использованию виде (jar-файлы), так и в исходниках
● Для подключения библиотеки (файл *.jar) достаточно создать папку libs в проекте (на том же уровне, что и папки src и res) и скопировать туда файл библиотеки
● Добавить ее в проект через меню Project -> Properties
● Если библиотека представлена в виде исходного кода, необходимо ее предварительно собрать
● Щелкнуть правой кнопкой по корневой папке проекта -> Export: -> Java -> Runnable JAR file ->Указать класс для запуска -> Указать место сборки -> Finish

Библиотеки совместимости:
Позволяют использовать возможности, появившиеся в какой-то версии ОС Android, на более ранних версиях платформы
Разработчик с одной стороны должен ориентироваться на новые возможности и уметь их использовать, а с другой – стараться сделать так, чтобы приложение работало на максимальном количестве устройств

При настройке обратной совместимости необходимо отредактировать файл манифеста, указав в нем минимальную версию Android SDK, которая необходима для запуска приложения, и основную (целевую) версию:

<uses-sdk android:minSdkVersion="7" android:targetSdkVersion="17" />

Примеры библиотек:
NineOldAndroids - для использования анимации, доступной в HoneyComb и выше.
ActionBarSherlock - использование компонента ActionBar в версиях меньше 4.0.

Библиотеки спец. назначения используются для игр, соц. сетей, статистики.
Yandex.Metrica for Apps — набор библиотек для сбора статистики использования мобильного приложения, информация об устройстве, информация о сессиях, информация об источнике перехода пользователя на страницу скачивания приложения, действия, выполненные пользователем в приложении.
FaceBook SDK - чтобы писать сообщения на стену, менять статус и т.д.

Прикладные библиотеки:
Различные библиотеки, предоставляющие дополнительные возможности:

- Работа с изображениями
- Работа с картами
- Парсинг HTML-страниц
- Построение графиков
- Многое, многое другое

Universal Image Loader - Работа с изображениями: многопоточная загрузка изображений, широкие возможности настройки и конфигурирования, кеширование загруженных изображений как в оперативной памяти, так и на карте, поддержка виджетов Android 2.0 и выше.

JSoup - для парсинга HTML-страниц.
Android Holo ColorPicker - выбор цвета с использованием цветового колеса.
MapNavigator - работа с картами Google Maps, определение маршрута.
AChartEngine - графики функций, диаграммы.

45.Коммуникационные возможности смартфонов (телефонная связь и работа с SMS): обзор классов.

Возможности мобильной связи

- Возможность совершения телефонных звонков
- Возможность получения и отправки SMS
- Возможность выхода в Интернет через мобильную сеть

Использование стандартных приложений

- Часто нет смысла изобретать велосипед (приложение на замену стандартному)
- Вызов стандартных активностей смартфона, предназначенных для выполнения нужных действий
- Если на устройстве имеется несколько вариантов таких приложений, операционная система предложит пользователю выбор

Особенности работы с планшетом

- Планшеты на Android часто имеют разъем для подключения SIM-карт
- Редко позволяют использование в качестве телефона
- Может быть возможность работы с SMS
- Использование мобильного Интернета
- Проверять тип устройства!

Использование эмуляторов для тестирования обработки звонков

- Нужно создать несколько экземпляров эмуляторов
- Первый эмулятор получит абонентский номер 5554, номера следующих последовательно увеличиваются на 2 (5556, 5558, …)
- После запуска эмуляторов на одном из них необходимо открыть панель набора номера, ввести номер второго эмулятора и сделать вызов

Вызов номера из приложения

- Приложения Android способны вызывать телефонные номера, но только через запуск наборной панели
- Вызов экрана набора номера осуществляется через специальный интент

- Можно вызвать стандартную панель набора номера или передать в этот экран номер конкретного абонента

Работа с SMS

- Для работы с SMS используется класс SMSManager
- Для создания экземпляра класса SMSManager используется метод getDefault()

```
SmsManager smsManager = SmsManager.getDefault();
```

Отправка SMS

Метод sendTextMessage:

```
public void sendTextMessage

(String destinationAddress, //номер получателя

String scAddress, //номер центра сообщений оператора

String text, //текст

PendingIntent sentIntent, //отчет об отправке

PendingIntent deliveryIntent) //отчет о доставке
```

Отчет об отправке

- Activity.RESULT_OK, если сообщение было отправлено успешно
- RESULT_ERROR_GENERIC_FAILURE, RESULT_ERROR_RADIO_OFF или RESULT_ERROR_NULL_PDU, если сообщение отправить не удалось

Настройка разрешений в манифесте

<uses-permission android:name="android.permission.RECEIVE_SMS">
</uses-permission> //получение

<uses-permission android:name="android.permission.READ_SMS" />

//чтение

<uses-permission android:name="android.permission.SEND_SMS">

</uses-permission> //отправка

Перехват SMS

- При получении SMS-сообщения система генерирует Broadcast Intent
- Для перехвата входящих сообщений используется BroadcastReceiver
- Интент-фильтр (в манифесте) для отбора получаемых сообщений

Перехват SMS

```
<receiver
android:name="com.androidexample.broadcastreceiver.IncomingSms">

   <intent-filter>

   <action android:name="android.provider.Telephony.SMS_RECEIVED" />

   </intent-filter>

</receiver>
```

Перехват SMS

Добавить в проект новый класс IncomingSms:

```
public class IncomingSms extends BroadcastReceiver
```

Создать SmsManager для работы с сообщениями:

```
final SmsManager sms = SmsManager.getDefault();
```

Создать в этом классе метод onReceive, обрабатывающий полученные сообщения:

```
public void onReceive(Context
context, Intent intent)
```

Перехват SMS

```
final Bundle bundle = intent.getExtras();
try {

    if (bundle != null) {

        final Object[] pdusObj = (Object[]) bundle.get("pdus");

        for (int i = 0; i < pdusObj.length; i++) {

            SmsMessage currentMessage = SmsMessage.createFromPdu((byte[]) pdusObj[i]);
            String phoneNumber = currentMessage.getDisplayOriginatingAddress();

            String senderNum = phoneNumber;
            String message = currentMessage.getDisplayMessageBody();
            Log.i("SmsReceiver", "senderNum: "+ senderNum + "; message: " + message);
           // Show alert
            int duration = Toast.LENGTH_LONG;
            Toast toast = Toast.makeText(context, "senderNum: "+ senderNum + ",
message: " + message, duration);
            toast.show();

        } // end for loop
      } // bundle is null

} catch (Exception e) {
    Log.e("SmsReceiver", "Exception smsReceiver" +e);

}
```

46.Сетевые соединения: класс ConnectivityManager и класс NetworkInfo.

Основные аспекты работы с сетью и файлами

- В целях безопасности одно приложение не может получить доступ к файлам другого. Однако подобное ограничение не распространяется на файлы, расположенные на SD-карте
- При разработке приложений следует придерживаться следующего правила: критичные для работы приложения файлы записываются в память устройства, а дополнительные, особенно имеющие большой объем, лучше выносить на карту памяти
- Для периодической синхронизации можно сделать в настройках опцию «Синхронизировать только через wi-fi»
- Следует жестко контролировать подключения и отключения от сети, время работы сервисов
- Следует использовать защищенные сетевые протоколы (например, HTTPS)

для передачи пользовательских паролей и другой конфиденциальной информации

Включение разрешения на работу с сетью в манифест (AndroidManifest.xml)

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.bignerdranch.android.photogallery"
android:versionCode="1"
android:versionName="1.0" >
<uses-sdk
android:minSdkVersion="8"
android:targetSdkVersion="15" />
<uses-permission android:name="android.permission.INTERNET" />
...
</manifest>
```

Сетевые соединения и способы подключения

Соединения

- WCDMA (3G)
- LTE (4G)
- wi-fi

Способы подключения

- HTTP,
- HTTPS,
- TCP/IP,
- сокеты

Основные аспекты реализации сетевой поддержки

- обеспечение конфиденциальности,
- учет потребления заряда батареи,
- проверка доступности сети.

Пакеты Android SDK для работы с сетью

- java.net
- java.io
- java.nio
- org.apache.*
- android.net
- android.net.http

- android.net.wifi
- android.telephony.gsm

Класс ConnectivityManager

- Мониторинг сетевых соединений (Wi-Fi, GPRS, UMTS и т.д.)
- Рассылка интентов при изменении статуса сетевых соединений
- Подключение к альтернативной сети, если соединение было потеряно
- Предоставляет API, который позволяет приложениям отправлять запросы на получение статуса сетевых соединений

Методы ConnectivityManager

- getActiveNetworkInfo () – получение информации об активном сетевом соединении
- getAllNetworkInfo () – получение информации о всех сетевых соединениях, поддерживаемых устройством
- getNetworkInfo (int networkType) – возвращает информацию о статусе текущего сетевого соединения

Управление сетевыми подключениями

Вызов менеджера соединений

```
String service = Context.CONNECTIVITY_SERVICE;
ConnectivityManager connectivity =
(ConnectivityManager) getSystemService(service);
```

Права на чтение/запись состояний сети для приложения

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE"/>
```

Класс NetworkInfo

- getType () – возвращает целочисленное значение, определяющее тип сети. Это может быть одна из констант TYPE_MOBILE, TYPE_WIFI, TYPE_WIMAX,

TYPE_ETHERNET, TYPE_BLUETOOTH или какое-то другое выражение, определенное в классе ConnectivityManager

- getSubtype () – возвращает значение, определяющее тип подсети
- getSubtypeName () – возвращает описательное имя типа подсети
- isAvailable () – проверяет доступность сети, в случае положительного ответа возвращает true
- isConnected () – возвращает true, если сетевое соединение установлено и через него можно передавать данные
- isConnectedOrConnecting ()– возвращает true в случае обнаружения установленного или устанавливаемого сетевого соединения
- isRoaming () – возвращает true, если мобильное устройство находится в роуминге

Определение статуса сети

```
// Get the active network information.
NetworkInfo activeNetwork = connectivity.getActiveNetworkInfo();
int networkType = networkInfo.getType();

switch (networkType) {
    case (ConnectivityManager.TYPE_MOBILE) : break;
    case (ConnectivityManager.TYPE_WIFI) : break;
    default: break;
}

// Get the mobile network information.
int network = ConnectivityManager.TYPE_MOBILE;
NetworkInfo mobileNetwork = connectivity.getNetworkInfo(network);

NetworkInfo.State state = mobileNetwork.getState();
NetworkInfo.DetailedState detailedState =
mobileNetwork.getDetailedState();
```

Выбор предпочтительной сети

Настройки предпочтительной сети:

- getNetworkPreference()
- setNetworkPreference()

```
int networkPreference = connectivity.getNetworkPreference();
connectivity.setNetworkPreference(NetworkPreference.PREFER_WIFI);
```

Если предпочтительная сеть недоступна или соединение с ней потеряно, то Android автоматически подключается ко второй сети. Для контроля доступности сети используется метод setRadio().

```
connectivity.setRadio(NetworkType.WIFI, false);
connectivity.setRadio(NetworkType.MOBILE, true);
```

Мониторинг сетевого трафика

Класс TrafficStats и методы данного класса для получения данных о трафика мобильного интернета:

- getMobileRxBytes()
- getMobileRxPackets()
- getMobileTxBytes()
- getMobileTxPackets()

Класс TrafficStats и методы данного класса для мониторинга сетевого трафика:

- getTotalRxBytes()
- getTotalRxPackets()
- getTotalTxBytes()
- getTotalTxPackets()

47.Использование Wi-Fi в Android: основные классы и их характеристики.

Управление wi-fi сетями

WifiManager в Android управляет услугой подключения WiFi:

- Настройка сетевых соединений Wi-Fi
- Управление текущим соединением Wi-Fi
- Сканирование точек доступа
- Монитор изменений WiFi-подключения

Мониторинг wi-fi подключения

Вызов wi-fi менеджера

```
String service = Context.WIFI_SERVICE;
WifiManager wifi = (WifiManager) getSystemService(service);
```

```
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
```

Мониторинг wi-fi подключения

Мониторинг и изменение состояния wi-fi

- Состояние сети:
  - WIFI_STATE_ENABLING - доступна,
  - WIFI_STATE_ENABLED - подключена,
  - WIFI_STATE_DISABLING - отключается,
  - WIFI_STATE_DISABLED - отключена,

- WIFI_STATE_UNKNOWN - неизвестна

```
if (!wifi.isWifiEnabled())
     if (wifi.getWifiState() != WifiManager.WIFI_STATE_ENABLING)
          wifi.setWifiEnabled(true);
```

Управление настройками Wi-Fi-соединения

- Класс Settings:
  - ACTION_WIFI_SETTINGS
  - ACTION_WIFI_IP_SETTINGS
  - ACTION_WIRELESS_SETTINGS
- Класс WifiManger
  - ACTION_PICK_WIFI_NETWORK

```
Intent intent = new Intent(Settings.ACTION_WIFI_SETTINGS);

startActivity(intent);
```

Характеристики сетевого подключения

Сеть Wi-Fi и ее точки доступа характеризуются специальными идентификаторами. Имя для идентификации сети называется идентификатором обслуживания сети или идентификатором SSID (Service Set Identifier).
Идентификатор текущей точки доступа -
BSSID (Basic Service Set Identifier).

Класс WifiInfo:

- getSSID()
- getBSSID()
- getRssi()
- getLinkSpeed()

Мониторинг параметров текущего подключения

После того, как подключаемся к точке доступа, используем getConnectionInfo из

WifiManager, чтобы получить информацию о сети:

- Возвращает объект "WifiInfo"

```
WifiInfo info = wifi.getConnectionInfo();

if (info.getBSSID() != null) {
    int strength = WifiManager.calculateSignalLevel(info.getRssi(), 5);
    int speed = info.getLinkSpeed();
    String units = WifiInfo.LINK_SPEED_UNITS;
    String ssid = info.getSSID();
    String cSummary = String.format("Connected to %s at %s%s. Strength
    %s/5",
    ssid, speed, units, strength);
}
```

IP-адресация

- ipAddress;
- gateway;
- dns1;
- dns2;
- netmask;
- serverAddress.

Сканирование точек доступа

- Используем WifiManager для сканирования точек доступа с помощью метода startScan()
- Поля класса startScan():
  - SSID , BSSID
  - capabilities
  - frequency
  - level
- Android транслирует результаты сканирования с помощью Интента

“SCAN_RESULTS_AVAILABLE_ACTION”

Сканирование точек доступа

```
// Register a broadcast receiver that listens for scan results.
registerReceiver(new BroadcastReceiver() {
     @Override
     public void onReceive(Context context, Intent intent) {
          List<ScanResult> results = wifi.getScanResults();
          ScanResult bestSignal = null;
          for (ScanResult result : results) {
               if (bestSignal == null ||
               WifiManager.compareSignalLevel(bestSignal.level,result.level)<0)
                    bestSignal = result;
          }
          String toastText = String.format("%s networks found. %s is
          the strongest.",
          results.size(), bestSignal.SSID);
          Toast.makeText(getApplicationContext(), toastText, Toast.LENGTH_LONG);
     }
}, new IntentFilter(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION));

// Initiate a scan.
wifi.startScan();
```

Конфигурация Wi-Fi-соединения

Конфигурацию сети определяет объект класса WifiConfiguration. Метод getConfiguredNetworks() возвращает список всех сетевых конфигураций в виде объекта List<WifiConfiguration>:

```
manager = (WifiManager)getSystemService(Context.WIFI_SERVICE);

List<WifiConfiguration> configs =
manager.getConfiguredNetworks();
```

Создание конфигурации WiFi

- Чтобы подключиться к WiFi сети, сеть должна быть создана и зарегистрирована:
    - Обычно пользователь создает настройки, но приложение может так же
- Параметры сети сохраняются как WifiConfiguration объект:
    - SSID (имя для идентификации беспроводной сети, например IPv4_KAIST)

- BSSID (MAC-адрес точки доступа)
- networkId (уникальный ID, который используется для идентификации параметров сети)
- priority (приоритет точки доступа)
- status (текущий статус: ENABLED, DISABLED, CURRENT)

Поддержка различных типов алгоритмов и протоколов безопасности

Методы класса WifiConfiguration для поддержки различных типов алгоритмов и протоколов безопасности:

- allowedProtocols()
- allowedAuthAlgorithms()
- allowedGroupCiphers()
- allowedKeyManagement()
- wepKeys()
- wepTxKeyIndex()
- allowedPairwiseCiphers()
- preSharedKey()

Создание конфигурации сети

```
WifiManager wifi = (WifiManager) getSystemService(Context.WIFI_SERVICE);
WifiConfiguration wc = new WifiConfiguration();

wc.SSID = "\"SSIDName\"";
wc.preSharedKey  = "\"password\""; // it should be in double quote “password”
wc.hiddenSSID = true;
wc.status = WifiConfiguration.Status.ENABLED;

// setting up WPA-PSK
wc.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.TKIP);
wc.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.CCMP);
wc.allowedKeyManagement.set(WifiConfiguration.KeyMgmt.WPA_PSK);
wc.allowedPairwiseCiphers.set(WifiConfiguration.PairwiseCipher.TKIP);
wc.allowedPairwiseCiphers.set(WifiConfiguration.PairwiseCipher.CCMP);
wc.allowedProtocols.set(WifiConfiguration.Protocol.RSN);

int res = wifi.addNetwork(wc); // the ID of the newly created network description
Log.d("WifiPreference", "add Network returned " + res );
boolean b = wifi.enableNetwork(res, true);
Log.d("WifiPreference", "enableNetwork returned " + b );
```

Управление сетями Wi-Fi

Используем Use WiFi Manager для управления параметрами беспроводных сетей и для управления подключениями к той или иной сети

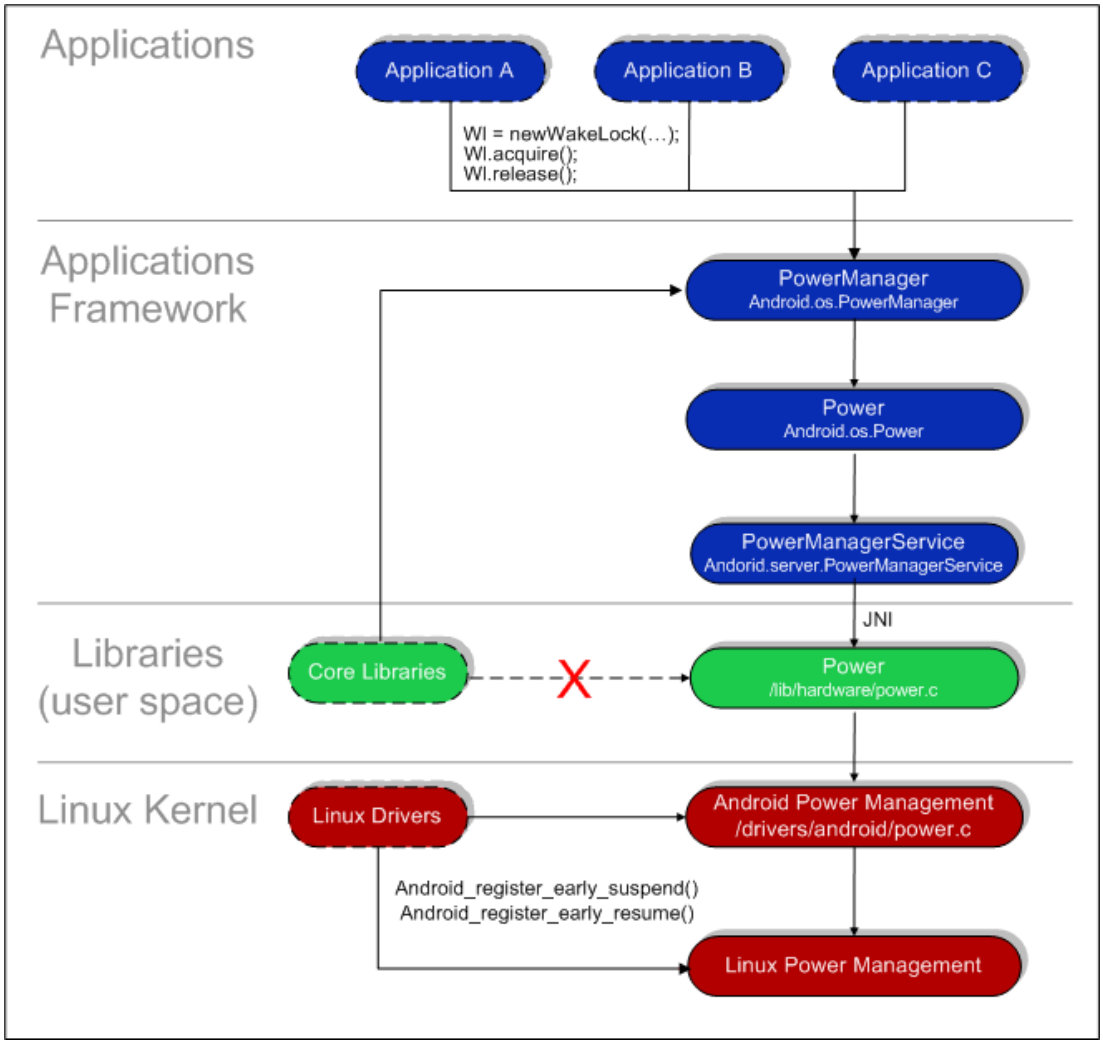
```
// Get a list of available configurations
List<WifiConfiguration> configurations = wifi.getConfiguredNetworks();

// Get the network ID for the first one.
if (configurations.size() > 0) {
    int netID = configurations.get(0).networkId;
    // Enable that network.
    boolean disableAllOthers = true;
    wifi.enableNetwork(netID, disableAllOtherstrue);
}
```

Управление энергосбережением смартфона

- Android поддерживает собственный механизм управления ресурсами и энергосбережением (поверх модели управления ресурсами, используемой в Linux):
    - Проверять не используются ли ресурсы процессора, если они не требуются приложению
- Android требует, чтобы приложения и сервисы запрашивали ресурсы CPU с помощью "Wakelocks»:
    - wakelock (блокировка сна) - это специальные события, препятствующие переходу устройства в спящий режим. Эти события могут генерироваться как операционной системой, так и пользовательскими приложениями.
    - Если нет активных wakelock, то ресурсы процессора использоваться не будут.

Управление энергосбережением


Applications
Application A
Application B
Application C
WI = newWakeLock(…);
WI.acquire();
WI.release();
Applications Framework
PowerManager
Android.os.PowerManager
Power
Android.os.Power
PowerManagerService
Andorid.server.PowerManagerService
JNI
Libraries (user space)
Core Libraries
Power
/lib/hardware/power.c
Linux Kernel
Linux Drivers
Android Power Management
/drivers/android/power.c
Android_register_early_suspend()
Android_register_early_resume()
Linux Power Management

Wakelock

Flag value	CPU	Screen	Keyboard
PARTIAL_WAKE_LOCK	On	Off	Off
SCREEN_DIM_WAKE_LOCK	On	Dim	Off
SCREEN_BRIGHT_WAKE_LOCK	On	BRIGHT	Off
FULL_WAKE_LOCK	On	Bright	Bright

```
// Acquire handle to the PowerManager service
PowerManager pm = (PowerManager)mContext.getSystemService(
                  Context.POWER_SERVICE);
// Create a wake lock and specify the power management flags for
screen, timeout, etc.
PowerManager.WakeLock wl = pm.newWakeLock(
PowerManager.SCREEN_DIM_WAKE_LOCK |   PowerManager.ON_AFTER_RELEASE,
TAG);
// Acquire wake lock
wl.acquire(); // ...
// Release wake lock
wl.release();
```

Фоновая передача данных

- Фоновая передача данных:
  - Wifilock + Wakelock (partial)

```
// http://developer.android.com/reference/android/net/wifi/WifiManager.WifiLock.html
WifiManager.WifiLock wifiLock = null;
PowerManager.WakeLock wakeLock = null;
// acquire
if (wifiLock == null) {
     WifiManager wifiManager = (WifiManager)
context.getSystemService(context.WIFI_SERVICE);
     wifiLock = wifiManager.createWifiLock("wifilock");
     wifiLock.setReferenceCounted(true);
     wifiLock.acquire();

          PowerManager powerManager = (PowerManager)
          context.getSystemService(context.POWER_SERVICE);
          wakeLock =
          powerManager.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK, "wakelock");
          wakeLock.acquire();

}
// release
if (wifiLock != null) {
     wifiLock.release();
     wifiLock = null;
     wakeLock.release();
     wakeLock = null;
}
```

Фоновая передача данных

Приложение может отслеживать изменения параметров фоновой передачи данных:

```
registerReceiver(
    new BroadcastReceiver() {
      @Override
      public void onReceive(Context context, Intent intent)
      // do something..
  },
   new IntentFilter(ConnectivityManager.
      ACTION_BACKGROUND_DATA_SERVICE_CHANGED)
    );
```

Фоновая передача данных

- Setting > Accounts & sync settings > настройка фоновой передачи данных
- Если фоновая передача данных отключена, то приложение не может передавать данные
- Использовать менеджер соединений для проверки параметров фоновой передачи данных:

```
boolean backgroundEnabled =
```

```
connectivity.getBackgroundDataSetting();
```

48.Использование Bluetooth в Android.

Этапы работы с Bluetooth:

- установка настроек bluetooth адаптера,
- поиск доступных для соединения устройств,
- установка соединения,
- передача данных

Обзор Android Bluetooth API

- BluetoothAdapter
- BluetoothDevice
- BluetoothSocket
- BluetoothServerSocket
- BluetoothClass
- BluetoothProfile
- BluetoothHeadset
- BluetoothA2dp
- BluetoothHealth
- BluetoothHealthCallback
- BluetoothHealthAppConfiguration
- BluetoothProfile.ServiceListener

Использование Bluetooth модуля

- Подключение API:
- import android.bluetooth.*;
- Определение разрешений в манифесте:
- <uses-permission android:name="android.permission.BLUETOOTH" />
- Права на изменение имени устройства:
- <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
- Создать экземпляр класса BluetoothAdapter
- BluetoothAdapter bluetooth= BluetoothAdapter.getDefaultAdapter();

Организация поиска доступных bluetooth устройств

- Сканирование или запрос списка спаренных устройств с помощью класса BluetoothAdapter.

- Bluetooth-устройство в пределах досягаемости отправляет на запрос данные о себе: имя, класс, свой уникальный MAC адрес.
- После установки соединения с внешним устройство отображается запрос на соединение. Если ответ положительный, то данные устройства сохраняются.

Требуется различать спаренные и соединенные устройства. Текущая реализация bluetooth API требует, чтобы устройства были спарены перед соединением.