

Вводная лекция

Лекция 1

1. Создание программно - аппаратных информационных комплексов;
2. Развитие и поддержка информационных технологий внутри компании.
Системное администрирование;
3. Сфера информационных корпоративных бизнес – решений.

Специалисты, необходимые для создания программно - аппаратных информационных комплексов

10

1. Младшие специалисты по программированию или инженеры (стажеры);
2. Программисты и инженеры по программному обеспечению;
3. Старшие (ведущие) инженеры и руководители отдела;
4. Руководитель проекта.

Специалисты, необходимые для развития и поддержки информационных технологий внутри компании

1. Специалист службы технической поддержки (helpdesk);
2. Системный администратор;
3. IT – менеджер;
4. IT – директор.

Специалисты, необходимые для сферы информационных корпоративных бизнес- решений

1. Консультант по внедрению и сопровождению ИТ (специалист по предметной области конкретного бизнес решения);
2. Бизнес – аналитик (специалист по предметной области конкретного бизнес решения);
3. Руководитель проекта по внедрению и сопровождению ИТ (специалист по предметной области конкретного бизнес решения)

Перечень дисциплин необходимых для подготовки специалистов в первой области применения ИТ

13

- 1.Программирование на языках низкого уровня (ASSEMBLER)(Императивная парадигма; программирования);
- 2.Программирование на языке высокого уровня(язык С)(Императивная парадигма программирования);
- 3.Программирование на WEB языках(язык PHP) (Императивная парадигма программирования);
- 4.Программирование на WEB языках(язык Python, JAVA, Ruby)(Объектно-ориентированная парадигма);
- 5.Программирование на WEB языках(язык PERL) (Сентенциальное программирование);
6. Язык запросов СУБД;
7. Теория алгоритмов;
8. Теория грамматик и языков программирования;

Перечень дисциплин необходимых для подготовки специалистов во второй области применения ИТ

- 1.Программирование на языках низкого уровня (ASSEMBLER)(Императивная парадигма программирования);
- 2.Программирование на языке высокого уровня(язык С)(Императивная парадигма программирования);
- 3.Программирование на WEB языках(язык PHP) (Императивная парадигма программирования);
- 4.Программирование на WEB языках(язык Python или JAVA)(Объектно-ориентированная парадигма);
- 5.Системное администрирование(программирование командных файлов, загрузчиков, скриптов автоматизации деятельности сопровождения программных продуктов, обработчиков прерываний; установка, настройка и сопровождение общего и специального программного обеспечения; установка, настройка и сопровождение информационного обеспечения;создание загрузочных носителей дистрибутивов;ведение учетной политики и политики безопасности в различных программных plataформах Создание и сопровождение кластерных систем);
- 6.Язык запросов СУБД.

Перечень дисциплин необходимых для подготовки специалистов в третьей области применения ИТ

15

- 1.Функциональное программирование (HASCELL)(Функциональная парадигма программирования);
- 2.Логическое программирование(язык PROLOG)(Логическая парадигма программирования);
- 3.Событийное программирование(язык C++) (Событийная парадигма программирования);
4. Автоматное программирование(C)(Автоматная парадигма программирования);
5. Нейронно-сетевое программирование(C)(Нейронно-сетевая парадигма программирования);
6. Язык запросов СУБД;
7. Теория вычислительных процессов(параллельное или многопотокое программиоание, сети Петри, операционные системы);

Перечень дисциплин необходимых для подготовки специалистов не ИТ специальностей

16

- 1.Программирование на языке высокого уровня(язык С)(Императивная парадигма программирования);
- 2.Программирование на WEB языках(язык PHP) (Императивная парадигма программирования);
- 3.Программирование на WEB языках(язык Python или JAVA)(Объектно-ориентированная парадигма);
- 4.Язык запросов СУБД.

1. Программное обеспечение:

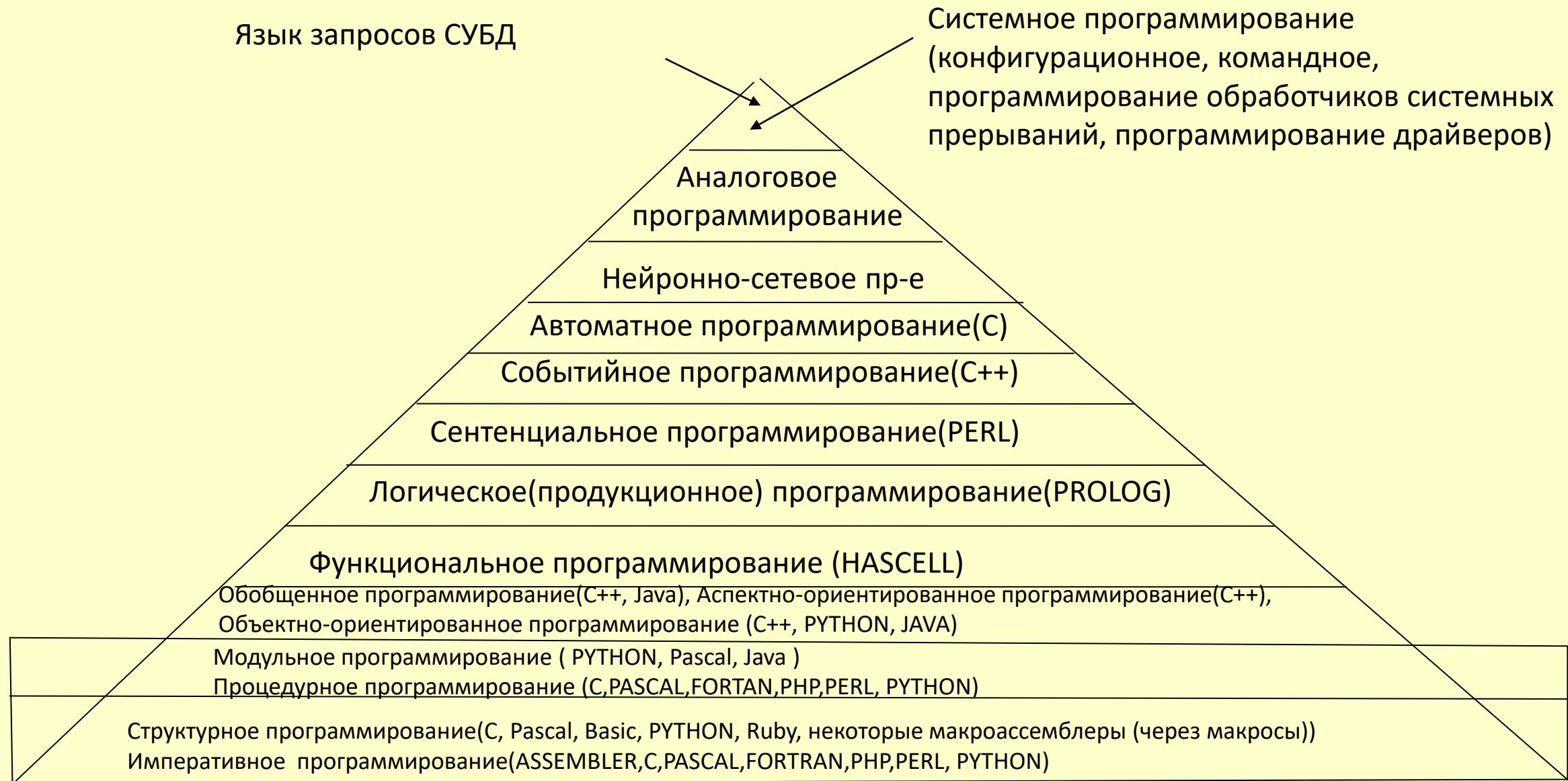
- **Общее программное обеспечение** – комплекс программ предназначенный для решения универсальных задач (операционные системы, антивирусные комплексы, браузеры, сетевые экраны, офисные приложения общего назначения, архиваторы)
- **Специальное программное обеспечение** – комплекс программ предназначенных для решения профессиональных, специфичных задач (программные комплексы систем автоматизированного проектирования, бухгалтерские и финансовые программные комплексы, программные системы для обработки растровой и векторной графики, программы для обработки видео и аудио информации, программные комплексы издательских систем, разнообразные программные системы для организации управления);
- **Алгоритмическое обеспечение** – совокупность алгоритмов формализованных в математической нотации и реализованных в программных продуктах в виде специальных библиотек.

2. Информационное обеспечение

- **Системы управления базами данных** – программные комплексы предназначенные для организации хранения, обработки и выдачи в необходимом виде конечному пользователю информации.
- **Базы данных** – совокупность связанных и несвязанных между собой объектов любой природы и описания их свойств

3. **Техническое обеспечение** – совокупность аппаратных элементов вычислительных систем необходимых для реализации программного и информационного обеспечения
4. **Организационное обеспечение** – совокупность организационных мероприятий, направленных на эффективное применение вычислительных систем

Современные парадигмы программирования



Императивное

программа = последовательность действий, связанных условными и безусловными переходами

процедурное

программа = последовательность процедур, каждая из которых есть последовательность элементарных действий и вызовов процедур, структурированных с помощью структурных операторов if, for и while

объектно-ориентированное

программа = несколько взаимодействующих объектов, функциональность (действия) и данные распределяются между этими объектами

функциональное

программа = система определений функций, описание того, что нужно вычислить, а как это сделать — решает транслятор; последовательность действий не прослеживается

продукционное (логическое)

программа = система определений и правил вида "условие => новый факт"

сентенциальное

программа = система правил вида "шаблон => трансформирующее действие"

событийное

программа = система правил вида "событие => новые события" + диспетчер событий

автоматное

программа = конечный автомат или автомат специального типа

Нейронно-сетевое

при программировании используется математический аппарат нейронных сетей

аналоговое

применяется для решения систем дифференциальных уравнений

системное

скриптовое применяется для организации сопровождения

Что нужно знать начиная изучать ОС Linux

В операционной системе Linux программное обеспечение поставляется в виде так называемых пакетов (package) — специальным образом подготовленных архивов, содержащих само программное обеспечение, его конфигурационные файлы, его данные и управляющую информацию.

Управляющая информация пакета включает контрольные суммы устанавливаемых файлов, зависимости устанавливаемого пакета от других пакетов, краткое описание пакета, сценарии установки, сценарии удаления пакета и прочие данные, необходимые менеджеру пакетов.

В примерах, приведенных в листингах, часто встречаются программы, которые, возможно, не будут установлены «по умолчанию», поэтому важно знать и понимать, как устроены подсистема управления установкой и удалением программного обеспечения — так называемые менеджеры пакетов и зависимостей.

Менеджер пакетов производит непосредственную установку и удаление пакетов программного обеспечения, а также ведет их учет в системе.

Вспомогательная «грязная» работа по подбору зависящих друг от друга пакетов, получению их из репозиториев (например, скачивание с FTP/HTTP-серверов), выбору правильных версий пакетов, определению правильного их порядка установки достается менеджеру зависимостей.

При помощи менеджера пакетов (листинг В1) всегда можно узнать имя пакета, в который входит та или иная установленная компонента операционной системы (например, /bin/date), или, наоборот, узнать список компонент, установленных из указанного пакета (например, coreutils).

Условно, можно выделить две ветви операционной системы Linux — ветвь debian, к которой относятся дистрибутивы W:[Debian] и W:[Ubuntu], и ветвь redhat, куда нужно отнести W:[RHEL], W:[CentOS] и W:[Fedora], в debian-ветви используется пакетный менеджер dpkg и построенные над ним менеджеры зависимостей apt, aptitude, synaptic и software-center, а в ветви redhat — пакетный менеджер rpm и основной менеджер зависимостей yum.

Листинг В1. Пакетный менеджер dpkg

```
bart@ubuntu:~$ which -a date
```

```
/usr/bin/date
```

```
/bin/date
```

```
bart@ubuntu:~$ dpkg -S /bin/date
```

```
coreutils: /bin/date
```

```
bart@ubuntu:~$ dpkg -L coreutils
```

```
/bin/chmod
```

```
/bin/chown
```

```
/bin/cp
```

```
/bin/date
```

```
/bin/dd
```

```
/bin/df
```

```
bart@ubuntu:~$ dpkg -s coreutils
```

Package: coreutils

Essential: yes

Status: install ok installed

Priority: required

Section: utils

Installed-Size: 7196

Maintained Ubuntu Developers <ubuntu-devel-discuss@lists.ubuntu.core>

Architecture: amd64

Multi-Arch: foreign

Version: 8.30-3ubuntu2

Pre-Depends: libacl1 (>= 2.2.23), libattr1 (>= 1:2.4.44), libc6 (>= 2.28), libselinux1 (>= .2.1.13)

Description: GNU core utilities

This package contains the basic file, shell and text manipulation utilities which are expected to exist on every operating system.

Specifically, this package includes:

arch base64 basename cat chcon chgrp chmod chown chroot cksum comm cp csplit cut date dd df dir dircolors dirname du echo env expand expr

Homepage: <http://gnu.org/software/coreutils>

Original-Maintainer: Michael Stone <mstone@debian.org>

Если при попытке выполнить ту или иную команду операционной системы Ubuntu Linux (это одна из причин, по которой иллюстрация в книге ведется именно с ее помощью) обнаружится, что нужный пакет с программным обеспечением не установлен, то при наличии доступа в Интернет можно тривиальным способом доустановить недостающие компоненты (листинг B2).

Листинг B2. Установка недостающего программного обеспечения

```
bart@ubuntu:~$ finger
```

Команда «finger» не найдена, но может быть установлена с помощью:

```
sudo apt install finger
```

```
bart@ubuntu:~$ sudo apt install finger
```

```
[sucJo] password for bart:
```

Чтение списков пакетов... Готово

Построение дерева зависимостей

Чтение информации о состоянии... Готово

Следующие НОВЫЕ пакеты будут установлены: finger

Обновлено 0 пакетов, установлено 1 новых пакетов, для удаления отмечено 0 пакетов, и 11 пакетов не обновлено.

Необходимо скачать 16,9 kB архивов.

После данной операции объём занятого дискового пространства возрастёт на 51,2 kB.

Пол:1 http://ru.archive.ubuntu.com/ubuntu eoan/universe and64 finger amd64
0.17-17 [16,9 kB]

Получено 16,9 kB за 0c (200 kB/s)

Подготовка к распаковке .../finger_0.17-17_and64.deb ...

Распаковывается finger (0.17-17) ...

Настраивается пакет finger (0.17-17) ...

Обрабатываются триггеры для man-db (2.8.7-3) ...

В редких случаях, когда нужно узнать, с каким, даже еще неустановленным, пакетом программного обеспечения поставляется тот или иной файл, может выручить утилита apt-file(1) (листинг В3).

В обратную сторону посмотреть список файлов, входящих в еще неустановленный пакет, можно этой же утилитой.

Листинг В3. в каком пакете файл (Файлы)?

```
bart@ubuntu:~$ apt-file search bin/7z
```

```
Finding relevant cache files to search ...E: The cache is empty. You need to run  
"apt-file update" first.
```

```
bart@ubuntu:~$ sudo apt-file update
```

```
Сущ:1 http://ru.archive.ubuntu.com/ubuntu eoan InRelease
```

```
Пол:2 http://ru.archive.ubuntu.com/ubuntu eoan-updates InRelease [97,5 kB]
```

```
Пол:3 http://ru.archive.ubuntu.com/ubuntu eoan-backports InRelease [88,8 kB]
```

Получено 91,9 МВ за 37с (2 515 kB/s)

Чтение списков пакетов... Готово

Построение дерева зависимостей

Чтение информации о состоянии... Готово

bart@ubuntu:~\$ apt-file search bin/7z

p7zip: /usr/bin/7zr

p7zip-full: /usr/bin/7z

p7zip-full: /usr/bin/7za

bart@ubuntu:~\$ apt-file show p7zip

p7zip: /usr/bin/7zr

p7zip: /usr/bin/p7zip

p7zip: /usr/lib/p7zip/7zr

p7zip: /usr/share/doc/p7zip/NEWS.Debian.gz

p7zip: /usr/share/doc/p7zip/README.Debian

p7zip: /usr/share/doc/p7zip/changelog.Debian.gz

p7zip: /usr/share/doc/p7zip/copyright

p7zip: /usr/share/man/man1/7zr.1.gz

p7zip: /usr/share/nan/nanl/p7zip.1.gz

Архитектура ОС Linux

Обзор внутреннего устройства

Операционная система (ОС) в общем и W:[Linux] в частности (рис. 1.1) представляет собой набор специализированных программных средств, обеспечивающих доступ потребителей (пользователей) к ресурсам (устройствам) и распределение последних между потребителями наиболее удобным и эффективным способом.

Под ресурсами в первую очередь понимают аппаратные средства, такие как центральные процессоры, память, устройства ввода-вывода, дисковые и прочие накопители и другие периферийные устройства.

Во вторую очередь к ресурсам относят такие «виртуальные» сущности (на рис. 1.1 не показаны), как процессы, нити, файлы, каналы и сокеты, семафоры и мьютексы, окна и прочие артефакты самой операционной системы, которые не существуют вне ее границ.

Как и во многих других операционных системах, в Linux выделяют два главных режима работы ее программных средств — ядерный режим (kernel mode), он же пространство ядра (kernel space), и пользовательский, внеядерный режим (user mode), или же пользовательское пространство (user space).

Основное различие этих двух режимов состоит в привилегиях доступа к аппаратным средствам — памяти и устройствам ввода-вывода, к которым разрешен полный доступ из режима ядра и ограниченный доступ из режима пользователя.

Совокупность работающих в ядерном режиме программ называют ядром, которое в Linux состоит из основы (или же остова) и присоединяемых к ней объектов — динамически загружаемых модулей.

Несмотря на компонентность, W:[Ядро Linux] относят к классу монолитных в силу того, что все его компоненты выполняются с одинаковыми (ядерными) привилегиями.

В классе микроядерных систем, наоборот, компоненты ядра работают с разными привилегиями: основная компонента — микроядро (планировщик процессов/нитей и менеджер памяти) — с наибольшими привилегиями, менеджер ввода-вывода, драйверы устройств, файловые системы, сетевые протоколы и пр. — с другими, обычно меньшими привилегиями (возможно даже, с привилегиями пользовательского режима).

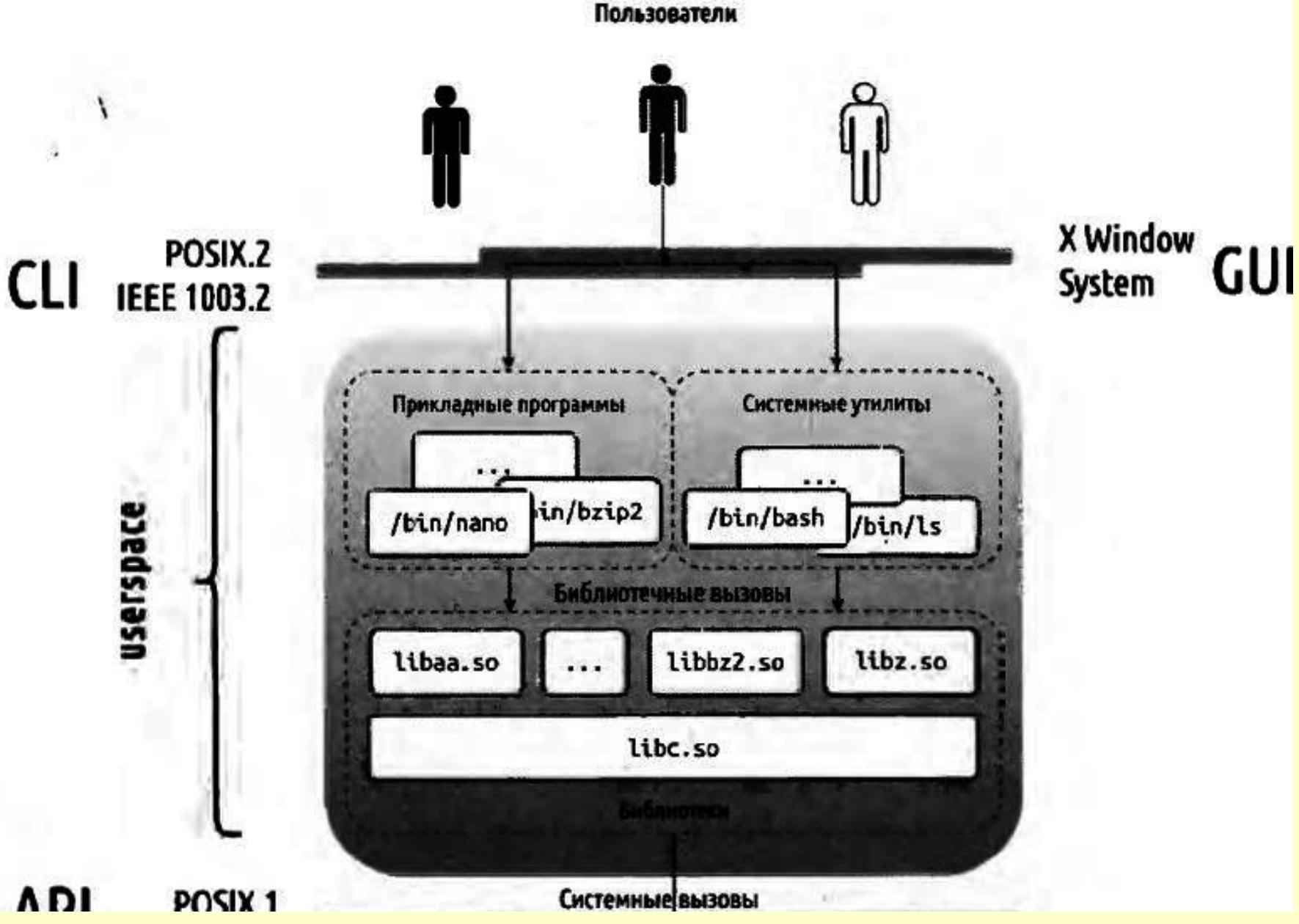


Рис. 1.1. Компоненты операционной системы Linux(часть 1)

API

**POSIX.1
IEEE 1003.1**

Системные вызовы

kernel space

**PCIe
SATA SCSI
USB**

Файловая подсистема

Виртуальный коммутатор ФС (VFS)

Файловые системы

Подсистема управления процессами

Планировщики процессоров

Процессы и нити

Подсистема ввода-вывода

Планировщики ввода-вывода

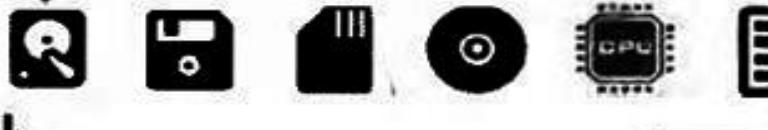
Драйвера устройств

Подсистема управления памятью

Страницочное распределение памяти

**Страницочный обмен
Страницочный кэш**

Виртуальная память



Устройства

Рис. 1.1. Компоненты операционной системы Linux(Часть 2)

Внеядерные компоненты: программы и библиотеки

Ядерные компоненты в основном обеспечивают решение задачи диспетчериизации (распределения) ресурсов между потребителями и предоставляют им базовый интерфейс доступа к ресурсам.

Решение задачи обеспечения удобства доступа реализуется компонентами внеядерного режима — библиотеками динамической и статической компоновки.

Функции операционной системы, реализуемые ядерными компонентами, доступны внеядерным компонентам посредством системных вызовов — специализированных наборов обращений для получения услуг ядра.

Системные вызовы выполняются в ядре, а вызываются при помощи основной внеядерной компоненты — библиотеки `libc.so` языка программирования W:[Си] (так исторически сложилось в силу того, что большая часть системных программных средств написана на этом языке).

Функции, реализуемые внеядерными компонентами, доступны посредством библиотечных вызовов и вызываются (выполняются) в самих библиотеках (например, алгоритмы сжатия информации в библиотеках `libz.so` и `Hbbz2.so`).

Ядерные компоненты: подсистемы управления процессами, памятью, вводом-выводом, файлами

Одна из основных задач ядра — распределение ресурсов между потребителями — вполне естественным образом приводит к тому, что среди ядерных компонент выделяют соответствующие (аппаратным ресурсам) менеджеры, так называемые подсистемы управления процессами, памятью, вводом-выводом и файловую подсистему.

Подсистема управления процессами распределяет время центральных процессоров (ЦП) между выполняющимися задачами (т. е. реализует W:[многозадачность]).

Она создает и уничтожает такие сущности, как процессы и нити, и организует одновременное (параллельное или псевдопараллельное) их выполнение при помощи планировщиков (scheduler), реализующих алгоритмы распределения процессорного времени.

Подсистема ввода-вывода распределяет доступ к устройствам ввода-вывода (УВВ) между процессами и предоставляет им унифицированные интерфейсы блочного, символьного и пакетного (сетевого) устройств.

Для устройств внешней памяти (дисковых или твердотельных накопителей, более медленных по сравнению с оперативной памятью) подсистема ввода-вывода организует W:[Виртуальную память] при помощи подсистемы управления памятью.

Подсистема управления памятью распределяет пространство оперативного запоминающего устройства (ОЗУ) между процессами при помощи механизма страничного отображения — выделяет (и высвобождает) процессам страничные кадры физической памяти и отображает на страницы их адресного пространства.

Кроме того, эта подсистема организует W:[виртуальную память] за счет механизма страничного обмена (W:[подкачка страниц]) — вытесняет неиспользуемые страницы процессов во внешнюю память и загружает их обратно по требованию при помощи подсистемы ввода-вывода.

Стоит отметить, что распределение ресурсов процессоров и устройств ввода-вывода происходит «во времени», т. е. в отдельные промежутки времени эти устройства выполняют операции только одного процесса или нити.

Наоборот, распределение ресурсов запоминающих устройств происходит «в пространстве», т. к. информация нескольких процессов одновременно размещается в разных их областях.

Файловая подсистема ядра предоставляет процессам унифицированный интерфейс файлового доступа к внешней памяти (внешним запоминающим устройствам, ВЗУ — магнитным дисковым, твердотельным накопителям и т. п.) и распределяет между ними пространство ВЗУ при помощи файлов и файловых систем.

Особенное назначение файловой подсистемы состоит еще и в том, что при помощи ее файлового интерфейса процессам предоставляется доступ и к другим подсистемам.

Так, доступ к устройствам ввода-вывода организуется посредством специальных файлов устройств (блочных и символьных), например к CD/DVD-накопителю — через файл `/dev/sr0`, а к манипулятору мыши — через `/dev/input/mouse0`.

Доступ к физической памяти и памяти ядра ОС организуется через файлы виртуальных устройств /dev/тmem и /dev/kтmem, а доступ процессов к страницам памяти друг друга — через файлы /ргос/PID/тmem псевдофайловой системы procfs.

Даже доступ к внутренним параметрам и статистике различных компонент ядра операционной системы возможен через файлы разнообразных псевдофайловых систем, например procfs, securityfs, debugfs и прочих, а к списку обнаруженных ядром устройств ввода-вывода — через файлы псевдофайловой системы sysfs.

Кроме всех вышеперечисленных задач, файловая подсистема, подсистема ввода-вывода и подсистема управления процессами в совокупности предоставляют процессам средства межпроцессного взаимодействия, такие как сигналы, каналы, сокеты и разделяемая память.

Трассировка системных и библиотечных вызовов

Для наблюдения за обращениями программ к услугам ядерных компонент операционной системы, т. е. за системными вызовами, служит утилита strace, предназначенная для трассировки — построения трасс выполнения той или иной программы.

В листинге 1.1 представлена трассировка программ whoami, hostname и pwd относительно системных вызовов geteuid, uname, getcwd и sethostname, где оказывается, что для получения и установки значения (сетевого) имени системы программа hostname использует разные системные вызовы uname и sethostname, при этом один из системных- вызовов является привилегированным и доступен только суперпользователю root

Листинг 1.1. Трассировщик системных вызовов strace

```
bart@ubuntu:~$ whoami
```

```
bart
```

```
bart@ubuntu:~$ hostname
```

```
ubuntu
```

```
bart@ubuntu:~$ pwd
```

```
/hone/bart
```

```
bart@ubuntu:~$ strace -fe uname,getcwd,geteuid,sethostname whoami
```

```
geteuid() = 1000
```

```
Bart
```

```
+++ exited with 0 +++
```

```
bart@ubuntu:~$ strace -fe uname,getcwd,geteuid,sethostname hostname
uname({sysname="Linux", nodename="ubuntu", ...}) =0
Ubuntu
```

```
+++ exited with 0 +++
```

```
bart@ubuntu:~$ strace -fe uname,getcwd,geteuid,sethostname pwd
getcwd("/hone/bart", 4096) = 11
/hone/bart
```

```
+++ exited with 0 +++
```

```
bart@ubuntu:~$ strace -fe uname,getcwd,geteuid,sethostname hostname Springfield
sethostname("sprlNgfleld", 11) = -1 EPERM (Операция не позволена)
hostname: you must be root to change the host name
+++ exited with 1 +++
```

В листинге 1.2 показана трассировка программы date относительно библиотечных вызовов fwrite или таких, в имени которых встречается строка time.

Оказывается, что эти библиотечные функции находятся в библиотеке языка Си, что весьма ожидаемо в силу внутреннего устройства операционной системы (см. рис. 1.1).

Листинг 1.2, Трассировщик системных вызовов ltrace

```
bart@ubuntu:~$ ltrace -x *time*+fwrite date
clock_gettime@libc.so.6(0, 0x7ffc8601a300, 1, 0x56056f3b04c0) = 0
localtime_r@libc.so.6(0x7ffc8601a230, 0x7ffc8601a240, 4, 269) = 0x7ffc8601a240
strftime@libc.so.6( unfinished ...)
strftime_l@libc.so.6(0x7ffc86019db0, 1024, 0x7ffc86019dab, 0x7ffc8601a240) = 5
<... strftime resumed> " \320\241\320\261", 1024, "%a", 0x7ffc8601a240) = 5
fwrite@libc.so.6(" \320\241\320\261", 4, 1, 0x7f3d7e8d56a0) = 1
strftime@libc.so.6( unfinished ...)
```

strftine_l@libc.so.6(0x7ffc86019db0, 1024, 0x7ffc86019dab, 0x7ffc8601a240) = 7
<... strftine resuned> " \320\275\320\276\321\217", 1024, " 9fl>,
0x7ffc8601a240) = 7 fwrite@libc.so.6("\320\275\320\276\321\217", 6, 1,
0x7f3d7e8d56a0) = 1

fwrite@libc.so.6("16", 2, 1, 0x7f3d7e8d56a0) = 1

fwrite@libc.so.6("20", 2, 1, 0x7f3d7e8d56a0) = 1

fwrite@libc.so.6("5", 1, 1, 0x7f3d7e8d56a0) = 1

fwrite@libc.so.6("9", 1, 1, 0x7f3d7e8d56a0) = 1

fwrite@libc.so.6("MSK", 3, 1, 0x7f3d7e8d56a0) = 1

fwrite@libc.so.6("2019", 4, 1, 0x7f3d7e8d56a0) = 1

C6 ноя 16 20:05:09 MSK 2019

+++ exited (status 0) +++

Интерфейсы прикладного программирования

Системные и библиотечные вызовы Linux, формирующие интерфейс прикладного программирования (API, application programming interface), соответствуют определенным промышленным спецификациям, в частности (практически идентичным друг другу) стандартам W: [POSIX], Portable Operating System Interface и SUS, W: [Single UNIX Specification], доставшимся «в наследство» от семейства операционных систем UNIX, членом которого Linux и является.

Стандарт POSIX условно делится на две части: POSIX.1, программный интерфейс (API) операционной системы, и POSIX.2, интерфейс командной строки (CLI) пользователя и командный интерпретатор.

Стандарт POSIX выпускается комитетом 1003 организации W:[IEEE], поэтому имеет формальное обозначение IEEE 1003, а части стандарта POSIX.1 и POSIX.2 формально обозначаются IEEE 1003.1 и IEEE 1003.2 соответственно.

Итоги

Совершенно очевидно, что в современной медицине без понимания анатомии живых организмов невозможно представить ни их терапию, ни хирургию.

В информационных технологиях абсолютно аналогичным образом разработка и эксплуатация программного обеспечения будут успешны только на основе понимания внутреннего устройства операционной системы.

Рисунок 1.1 изображает эдакий «рентгеновский снимок» внутренностей операционной системы Linux, подробная анатомия которой шаг за шагом и раскрывается в последующих материалах.

Пользовательское окружение ОС Linux

Командный интерфейс

Основным интерфейсом взаимодействия между ЭВМ и человеком в классической операционной системе W:[UNIX] был единственно возможный, диктуемый аппаратными устройствами ее времени *командный интерфейс*. Называемый сегодня интерфейсом командной строки (Command Line Interface, W:[CLI]), он в неизменном виде сохранил все свои элементы — понятие *терминала*, двусторонний попеременный диалог при помощи клавиши Enter, управляющие символы и клавишу для их набора.

Терминал является оконечным (англ, *terminal*) оборудованием, предназначенным для организации человека-машинного интерфейса. Обычно он состоит из устройств вывода — принтера или дисплея, и устройств ввода — клавиатуры, манипулятора «мышь» и пр.

Алфавитно-цифровой терминал позволяет вводить и выводить символы из некоторого заданного набора (например, семибитной кодировки ascii или другого набора символов charsets, состоящего из букв алфавита, цифр, знаков препинания, некоторых других значков, и символов специального назначения для управления самим терминалом — управляющих символов).

Ранние, печатающие терминалы, представляли собой телетайпы W:[телетайп] (телепринтеры W:[teleprinter]), которые печатали символы из фиксированного набора на ленте или рулоне бумаги — слева направо, сверху вниз.

Управляющие символы использовались для управления перемещением печатающей головки справа налево (символ BS), возврата головки к началу строки (символ CR), прокрутки рулона бумаги (символ LF) и пр.

Дисплейный терминал (видеотерминал на основе электронно-лучевой трубки) в упрощенном своем режиме эмулирует поведение печатающего терминала: поворачивающийся рулон бумаги — при помощи скроллинга изображаемых строк снизу вверх, а перемещающуюся вдоль строки печатающую головку — при помощи курсора.

В расширенном режиме видеотерминал используется как матрица символов, например в 24 строки по 80 символов в строке, и позволяет выводить символы в произвольное место матрицы, задавать символам стиль изображения, как то: мерцание, жирность, инвертирование, подчеркивание и цвет, — и даже менять шрифты символов терминала.

Для управления курсором, его позиционирования, смены стиля изображения символов и прочих возможностей видеотерминала применяются управляющие символы и управляющие последовательности.

Двусторонний попеременный диалог (рис. 2.1) командного интерфейса между пользователем и операционной системой представляет собой процесс ввода команд .

О пользователем посредством клавиатуры и получения результата их выполнения © на бумаге или дисплее алфавитно-цифрового терминала.

Другие пользователи

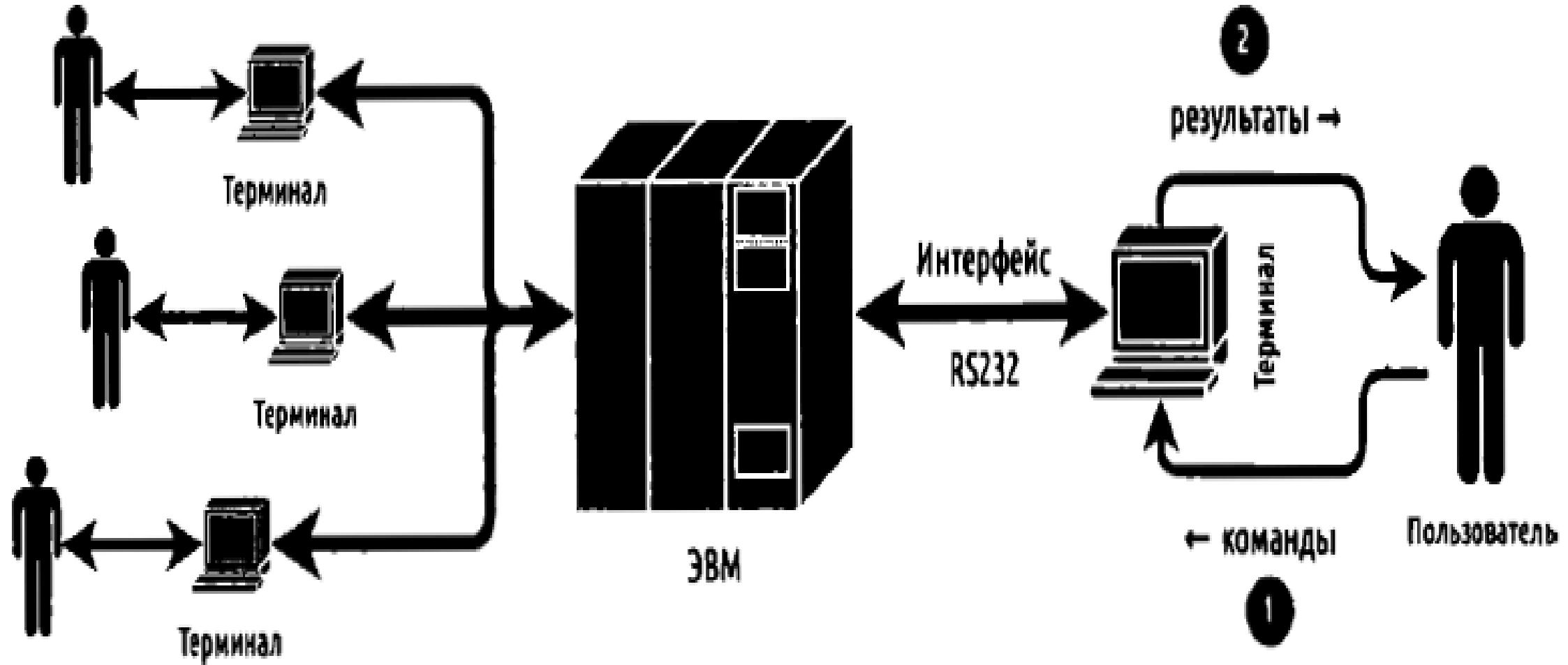


Рис. 2.1. Терминалы и командный интерфейс

В начале сеанса работы в многопользовательской среде операционной системы

пользователь должен произвести регистрацию (logging in) себя в системе (обычно говорят

«произвести вход» в систему) при помощи предъявления имени своей учетной записи (login) и соответствующего ему пароля (password, буквально – пропускное pass слово word) (рис. 2.2).

Процедура регистрации начинается с заставки операционной системы и приглашения к вводу имени учетной записи , в ответ на которое пользователь вводит имя своей учетной записи Ф. Затем, в ответ на приглашение к вводу пароля , пользователь вводит пароль , и при этом на алфавитно-цифровых терминалах никакие символы не изображаются. При положительном исходе регистрации пользователь получает сообщение о последней (last) успешной регистрации, сообщение дня и приглашение командного интерпретатора.

Передача управления от пользователя к операционной системе на каждом шаге диалога происходит при помощи нажатия клавиши Enter, а передача управления в обратную сторону — при помощи приглашений к вводу регистрационного имени, пароля, командного интерпретатора и пр.

Приглашение командного интерпретатора исторически состояло из символа \$ или символа ^, а при регистрации под учетной записью администратора — из символа #.

Позднее приглашение развилось в finn@ubuntu:~\$ и состоит теперь из имени зарегистрировавшегося пользователя finn, собственного имени компьютера ubuntu, условного имени домашнего каталога пользователя, обозначенного символом ~, и «классического» символа приглашения \$.

Сеанс командного интерфейса пользователя продолжается двусторонним попеременным диалогом с командным интерпретатором, где пользователь вводит команды и получает результаты их выполнения

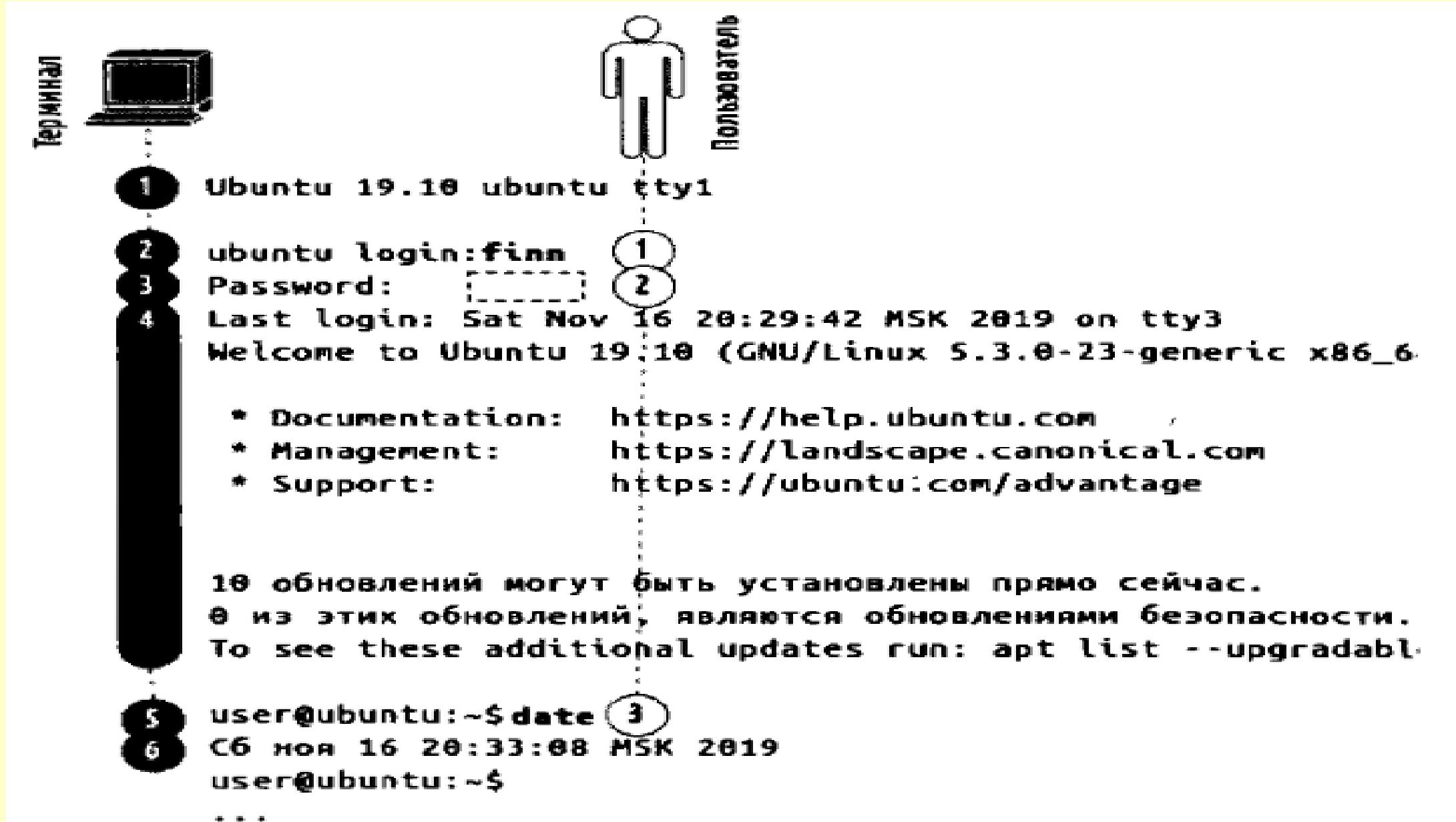


Рис 2.2 Регистрация пользователя в системе

Виртуальные терминалы

На текущий момент времени многопользовательские системы с настоящими физическими терминалами, подключенными посредством интерфейса RS232 и его драйвера `ttyS` к большой ЭВМ, — экзотическая редкость.

На рабочих станциях для взаимодействия с пользователем используются стандартные клавиатура, видеоадаптер и монитор, формирующие так называемую `W:[консоль]`, которая используется драйвером виртуальных терминалов для эмуляции нескольких физических терминалов.

Узнать имя текущего терминала (а точнее — имя специального файла устройства терминального драйвера, см. листинг 2.1), на котором выполнен вход в систему, позволяет команда `tty`, а список всех терминальных входов пользователей — команды `users`, `who` и `w`.

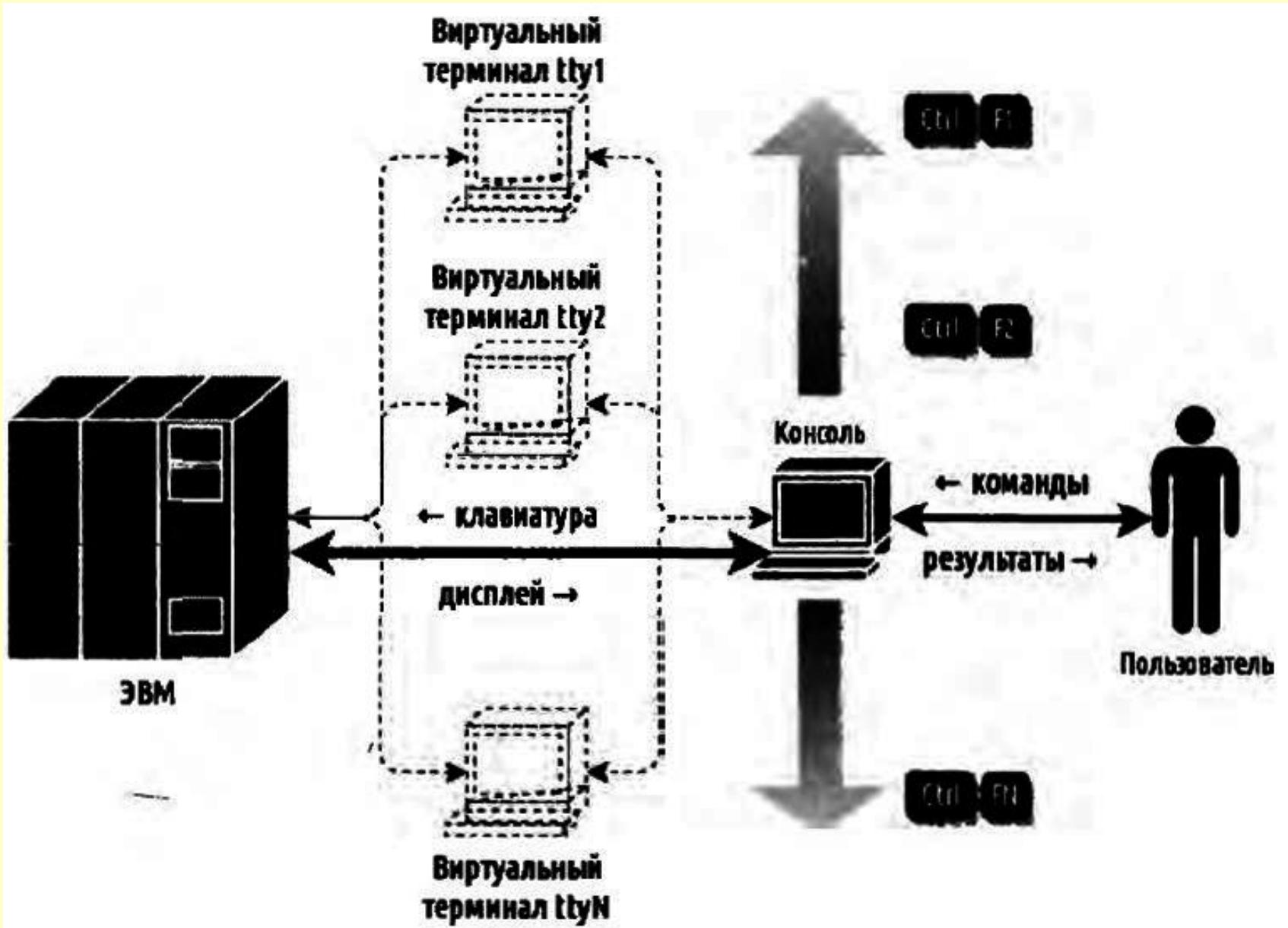


Рис. 2.3. Виртуальные терминалы

Листинг 2.1. Утилиты tty, users, who и w

```
ftnn@ubuntu:~$ tty
```

```
/dev/tty1
```

```
finn@ubuntu:~$ users
```

```
bubblegum flnn iceking jake jake marceline
```

```
finn@ubuntu:~$ who
```

```
iceking pts/0 2019-11-16 10:46 (176.10.35.129)
```

```
bubblegum tty5 2019-11-16 10:46
```

```
marceline tty3 2019-11-16 10:46
```

```
finn ttyl 2019-11-16 10:47
```

```
jake :0 2019-11-16 10:47 (:0)
```

```
jake pts/4 2019-11-16 22:09 (:0)
```

Драйвер виртуального терминала позволяет переключаться между эмулируемыми терминалами при помощи сочетания клавиш Ctrl F1 Ctrl F12 (первые двенадцать терминалов из 63 возможных), Alt <- для переключения на предыдущий, Alt -> для переключения на следующий виртуальный терминал.

При переключении из графического виртуального терминала на другой виртуальный терминал необходимо добавлять к сочетанию еще и клавишу Ctrl, т.к. сочетания с клавишей Alt

востребованы самим графическим интерфейсом, например Alt F1 закрывает активное окно. Таким образом, для переключения из графического на третий виртуальный терминал используется сочетание Alt Ctrl F3. Также драйвер виртуальных терминалов позволяет листать буфер вывода виртуального терминала при помощи сочетаний Ctrl PgU и Ctrl PgDN (к сожалению, после переключения терминалов буфер пропадает)

Как и любым другим, драйвером виртуальных терминалов можно управлять при помощи специально предназначенных команд, например, программа chvt позволяет переключаться на заданный терминал по его номеру, а команда программа setfont — загружать шрифты, формирующие начертания алфавитно-цифровых знаков (см. листинг 2.12).

Псевдотерминалы

При работе в оконной системе X Window System используются графические терминалы, тогда как для командного интерфейса требуется алфавитно-цифровой терминал.

В этом случае (рис. 2.4) он эмулируется при помощи драйвера псевдотерминала pty (pseudo tty) и приложения-посредника — эмулятора терминала (например, xterm или gnome-terminal), который связывает действительный обмен в графическом окне с мультиплексором псевдотерминалов ptmx (pseudoterminal multiplexer), а тот, в свою очередь, присоединен драйвером к подчиненному псевдотерминалу pts (pseudoterminal slave) командного интерфейса.

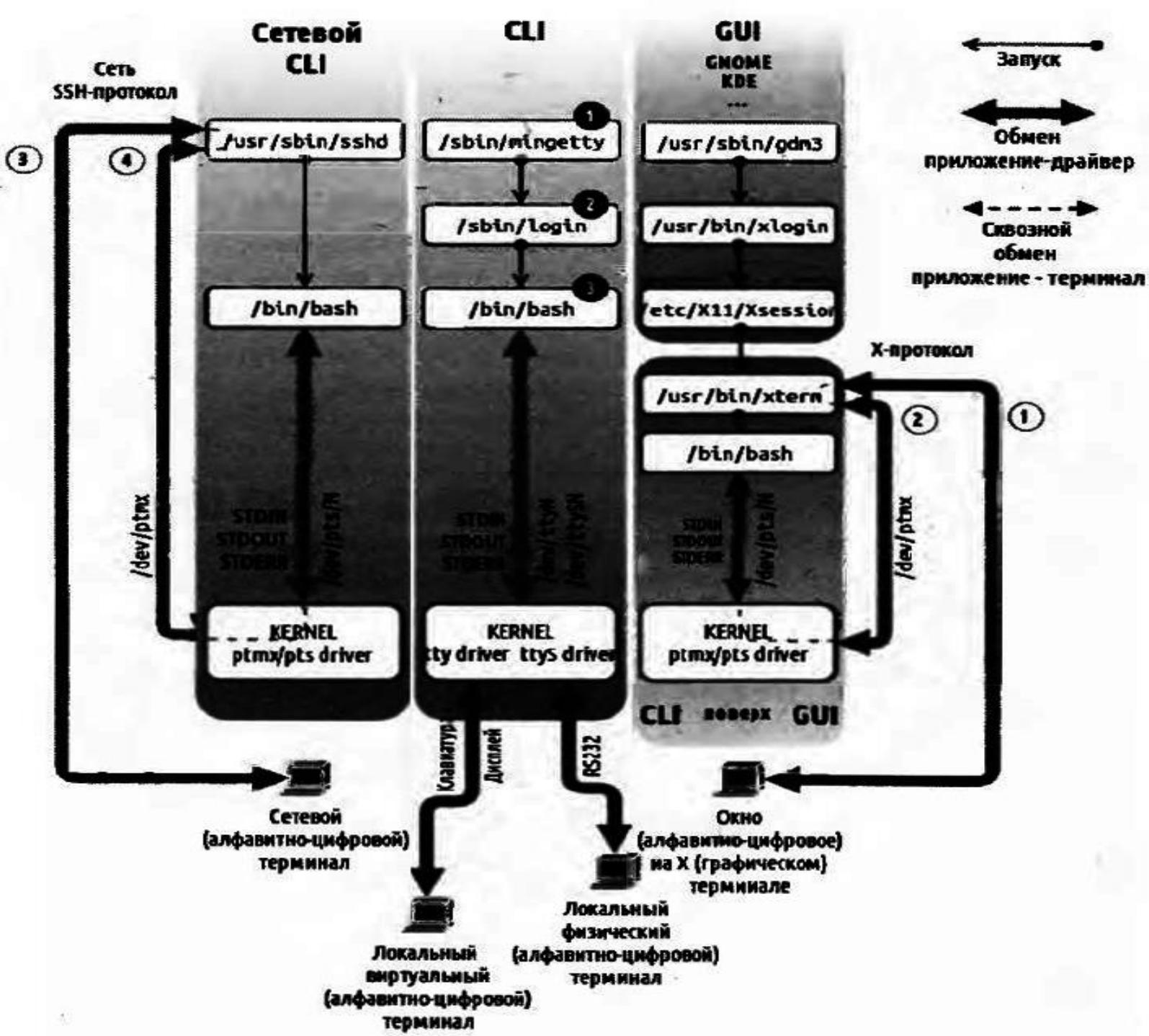


Рис. 2.4.
вдотерминалы

Аналогично, при подключении удаленного алфавитно-цифрового терминала посредством протоколов удаленного доступа (например, SSH) приложение-посредник (например, демон сетевой службы sshd, связывает действительный обмен в сетевом соединении с мультиплексором псевдотерминалов.

Таким образом, командный интерпретатор и запускаемые им программы работают с воображаемым псевдотерминалом так, как будто окно графического приложения или средства удаленного доступа являются настоящим физическим дисплеем и настоящей физической клавиатурой настоящего терминала.

В примере из листинга 2.1 пользователь finn зарегистрирован в системе на первом виртуальном терминале tty1, пользователь jake зарегистрирован на псевдотерминале pts/4 в эмуляторе терминала оконной системы, а пользователь iceking зарегистрирован при помощи удаленного доступа на псевдотерминале pts/0.

Нужно заметить, что на этапе входа пользователя в систему посредством алфавитно-цифрового терминала (см. средний фрагмент под заголовком CLI на рис. 2.4) последовательно запускаются: обработчик терминалов , обработчик аутентификации и авторизации пользователей ©, а затем командный интерпретатор , например bash.

Именно getty предъявляет пользователю (см. рис. 2.2) заставку операционной системы и приглашение к вводу имени пользователя, а login - приглашение к вводу пароля, сообщение о последнем успешном входе и сообщение дня.

Аналогичные процессы происходят при любом Входе пользователя в систему, например через псеводтерминалы «графического» или «сетевого» доступа (см. левый и правый фрагменты на рис. 2.4). В любом случае после аутентификации и авторизации основной программой (и первой в сеансе пользователя), интерпретирующей вводимые пользователем команды, является командный интерпретатор.

Управляющие символы

При вводе с терминала управляющие символы служат командами драйверу терминала и в большинстве своем генерируются при помощи сочетания клавши (отсюда ее название control – управление) с одной из алфавитно-цифровых клавиш.

В отдельных случаях управляющие символы генерируются специально предназначенными для этого клавишами, например Enter, Tab или Backspace.

Нужно заметить, что при работе с диалоговыми программами ^C или ^\ завершит выполняющуюся программу (at), не дав ей выполнить свое основное действие, или вообще будет проигнорирован (ftp, mail).

Именно символ завершения ввода (eof, end of file) сообщает драйверу о нежелании больше вести диалог с программой (который в свою очередь сообщит программе об отсутствии для нее вводимых данных).

В очень редких случаях, возможно, потребуется ввести сам управляющий символ, например ^C, ^\ или ^D, непосредственно в выполняющуюся на терминале программу, что невозможно сделать соответствующими клавиатурными комбинациями, потому как управляющие символы будут поглощены драйвером терминала, что приведет к завершению программы, в которую вводятся символы.

Для отмены (экранирования) специального назначения управляющих символов в пользу его непосредственного (литерального) значения служит управляющий символ (literal next) \next (^V), сигнализирующий драйверу терминала об отмене специального назначения следующего за ним символа

Управляющие последовательности

В расширенном режиме видеотерминалов W:[VT52], W:[VT100], W:[VT220] появилась возможность вывода символов в произвольное место экрана и использования полужирного, затемненного, негативного, подчеркнутого и других начертаний символов.

Возможности ввода дополнились функциональными клавишами, клавишами перемещения курсора, дополнительной клавиатуры и пр.

Для этого потребовались дополнительные управляющие символы, которые не поместились в кодировку ascii(7), потому терминалы стали использовать управляющие последовательности символов console_Codes, предваряемые управляющим символом ESC с кодом 6x1B.

Так, например, последовательность ESC # 8 вызовет визуальный тест выравнивания краев терминала, заполнением буквой Е всех строк и столбцов, ESC сбросит терминал в исходное состояние, ESC [In включит полужирное начертание, ESC [2 m – затемненное начертание, ESC [4 m – подчеркивание, а ESC [6 m вернет стандартное начертание символов.

Управляющие символы и их последовательности являются обычными байтами и при литеральном вводе с терминала могут быть сохранены в файл (листинг 2.9), например, при помощи команды tee.

При последующем выводе в на терминал, например при помощи команды cat, будут задействованы соответствующие расширенные возможности. Побайтное содержимое файла можно при этом увидеть на терминале посредством восьмеричного od или шестнадцатеричного дампа hexdump, hd.

Многие «дополнительные» клавиши современных терминалов, такие как функциональные F1...F12, клавиши управления курсором, скроллингом PgUp PgDn и пр., генерируют, управляющие последовательности, которые обрабатываются, например, библиотекой readline и используются для редактирования командной строки.

Несмотря на стандартизацию управляющих последовательностей, разные терминалы все же имеют различия, поэтому в операционной системе появились базы данных с описанием свойств и управляющих последовательностей терминалов terminfo и termcap.

Узнать ESC-последовательности можно при помощи команды infocmp, а вывести их на терминал — включить соответствующий режим — при помощи команды tput.

Примером простейшей программы, использующей управляющие последовательности для форматирования символов при выводе на экран, является утилита просмотра справочных страниц man.

В качестве более изощренных программ, использующих управление расширенными возможностями терминала, можно привести less, nano, mc, многие из которых используют для этого библиотеку ncurses.

А самым экстремальным примером использования управляющих последовательностей терминалов является W:[ASCII-графика] и W: [ANSI-графика], реализующаяся библиотеками aalib и caca, при помощи которых на алфавитно-цифровом (!) терминале можно просматривать видеофильмы (листинг 2.12), например, при помощи видеоплееров mplayer или mpv, поддерживающего эти библиотеки.

Листинг 2.12. Просмотр видео на алфавитно-цифровом терминале

```
finn@ubuntu:~$ setfont Unil-VGA8
```

```
finn@ubuntu:~$ mpv --quiet --vo=caca  
https://www.youtube.com/watch?v=Zo7_00W3GzA
```

```
flnn@ubuntu:~$ youtube-dl --exec 'nplayer -quiet -vo aa:din:bold:reverse'  
https://mmw.youtube.com/watch?v=Zo7_00W3CzA
```

Основной синтаксис командной строки

Основу интерфейса командной строки UNIX составляет командный интерпретатор (КИ), являющийся первой и главной программой, запускаемой в сеансе пользователя.

Двусторонний попеременный диалог с командным интерпретатором начинается с приглашения , в ответ на которое пользователь вводит команду, отправляя ее на выполнение управляющим символом LF 0x0A, получает результат ее исполнения на терминале и новое приглашение, сигнализирующее о готовности КИ к исполнению очередной команды (рис. 2.5).

Многие другие диалоговые программы, например lftp, также будут придерживаться синтаксиса и соглашений, принятых в языке командного интерпретатора.

Базовый синтаксис (подчиняющийся второй части стандарта W:[POSIX]) языка любого командного интерпретатора на самом деле достаточно прост и напоминает язык, близкий к естественному.

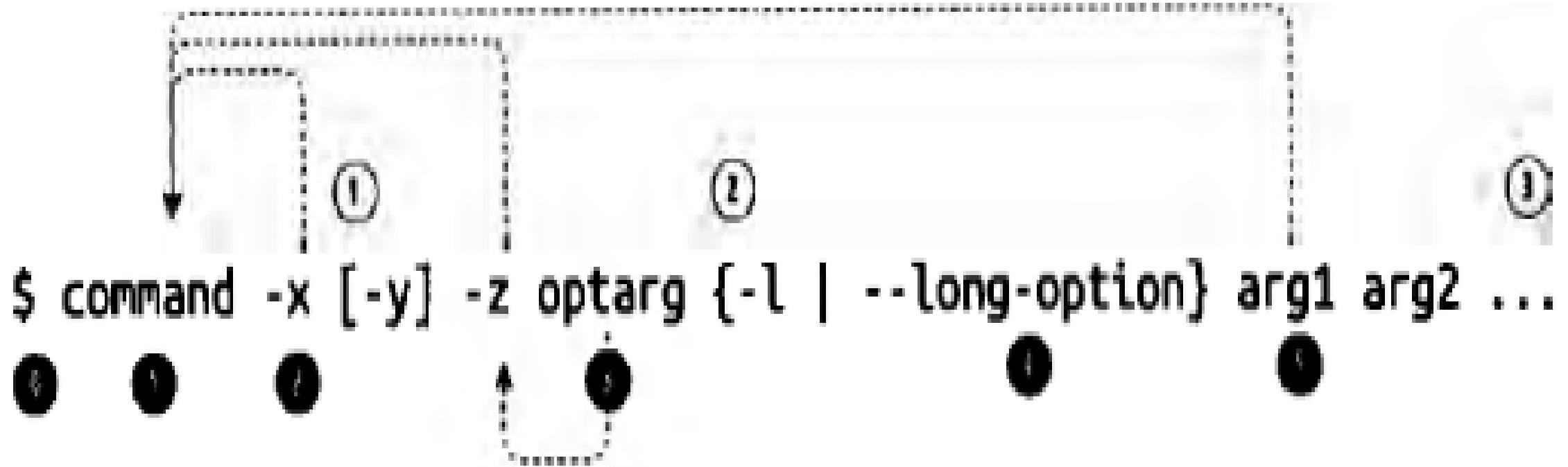


Рис. 2.5. Основной синтаксис командной строки

Например, гп -f -R Изображения Музыка переводится на человеческий как «удалить (rm) без шума и пыли (-f) и со всеми потрохами (-R) каталоги Изображения и Музыка».

Принято говорить, что команда состоит (см. рис. 2.5) из лексем (лексических элементов), разделенных пробельными символами — пробелами SP 0x20 и табуляциями HT 0x09 — в любом количестве и сочетании.

Первая лексема — это название команды, за которой следуют ее параметры: сначала опции (они же — ключи, они же — модификаторы) без аргументов и/или с аргументами, а в конце — аргументы самой команды.

Название команды — это глагол, указывающий, ЧТО делать; опции — это наречия и прочие части речи, объясняющие, КАК это делать; и наконец, аргументы задают то, с ЧЕМ это делать.

В разнообразной документации при описании синтаксиса команды, ее опций и аргументов принято использовать квадратные скобки для указания необязательности опции или аргумента, фигурные скобки и вертикальную черту для указания выбора из вариантов и многоточие для указания повторяемости.

Получив команду, интерпретатор определяет, является ли она псевдонимом, встроенной в интерпретатор, или реализуется внешней программой, подлежащей запуску (листинг 2.13).

Листинг 2.13. утилиты which и type

```
flnn@ubuntu:~$ which date
```

```
/usr/bin/date
```

```
finn@ubuntu:~$ type date
```

```
date является /usr/bin/date
```

```
finn@ubuntu:~$ type -a ls
```

```
ls - это псевдонимом для «ls -*color=auto»
```

```
ls является /usr/bin/ls
```

```
ls является /bin/ls
```

```
finn@ubuntu:~$ type -a pwd
```

```
pwd - это встроенная команда bash
```

```
pwd является /usr/bin/pwd
```

```
pwd является /bin/pwd
```

```
finn@ubuntu:~$ type which
```

```
для which вычислен хэш (/usr/bin/which)
```

```
finn@ubuntu:~$ type type
```

```
type - это встроенная команда bash
```

При наличии нескольких вариантов команды наивысший приоритет имеют псевдонимы, наименьший — внешние команды.

Подстановку псевдонимов можно увидеть, включив трассировку выполнения команд интерпретатора при помощи команды set (листинг 2.14).

Листинг 2.14, ПидЛановка псевдонимов

```
finn@ubuntu:~$ type ls
```

ls - это псевдонимом для «ls --color=auto»

```
finn@ubuntu:~$ set -x
```

```
ftnn@ubuntu:~$ ls -ASs
```

```
+ ls --color=auto -ASs
```

итого 36

```
12 examples.desktop 4 .bashrc 4 .profile 4 .lessht  
4 .cache 4 .bash_history 4 .bash_logout
```

Опции командной строки

В истории развития операционной системы UNIX программы использовали разные способы задания своих опций:

- ◆ односимвольные, например ls -a -l (что эквивалентно ls -l -a или ls -al или ls -la);
- ◆ многосимвольные, например find /var -xdev;
- ◆ длинные, например ps --help;
- ◆ с аргументами, например kill -n 15 1, или kill -nl5 1, или du -B И, или du -BM, или find /etc -type d, или даже ls --sort=size;
- ◆ «нестандартные», например set +x, tar czf tar.tgz ~ или dd if=/dev/dvd of=dvd.iso.

Знак «минус», предваряющий опции, естественно, используется для того, чтобы отличать их от аргументов.

Среди прочих он был выбран потому, что редко встречается как первый символ в аргументах команд (в качестве которых зачастую выступают имена файлов), и еще потому, что на терминале классической UNIX набор более логичного знака «плюс» (что могло бы означать «включить», «активировать») требовал достаточных усилий по нажатию клавиши Shift.

В результате получилось, что, например, в команде set опция x (execution trace) в форме -x включает, а +x выключает трассировку выполнения команд.

В тех редких случаях, когда аргумент команды все же начинается с символа «минус» и тем самым похож на опцию (представим, что нужно выполнить действие над файлом с именем -=filename=-), специальная опция -- сигнализирует о конце списка опций (листинг 2.15), за которым следуют лексемы, обязанные расцениваться как аргументы вне зависимости от их написания.

Листинг 2.15 Конец списка опций

```
finn@ubuntu:~$ stat -=ftlename=-
```

stat: неверный ключ - «=»

По команде «stat --help» можно получить дополнительную информацию.

```
finn@ubuntu:~$ stat -- -=filename=-
```

Файл: -=filename=-

Размер: 0 Блоков: 0 Блок В/В: 4096 пустой обычный файл

Устройство: fc00h/64512d Inode: 26870044 Ссылки: 1

Доступ: (0600/-тм.....)Uid: (1000/ finn) Ctd: (1000/ flnn)

Доступ: 2019-11-17 10:43:36.520984570 +0300 Модифицирован: 2019-11-17
10:43:36.520984570 40300 Изменён: 2019-11-17 10:43:36.520984570 40300

Создан: -

Короткие, односимвольные опции (например, -l -a) без своих аргументов издревле можно было объединять в группы (-la или -al), однако их в этом случае сложно отличать от многосимвольных (-xdev) или односимвольных, склеенных со своими аргументами (-BM).

Поэтому позже появились длинные (в так называемом, GNU- стиле) опции, обозначаемые двумя знаками «минус», позволяющие навести некоторый порядок в виде --block-size=M вместо -BM или, предположим, --dont-descent вместо -xdev.

Справочные системы

Используемые в Linux электронные справочные системы (online help) являются логичным следствием как его родства с семейством операционных систем UNIX — страницы руководства man (manual pages), так и принадлежностью к свободному программному обеспечению под эгидой движения GNU — справочная система info.

Следует отметить, что понятие online в контексте справочных систем вовсе не означает их доступность через Интернет, как это часто, но ошибочно воспринимается сегодня.

В рассматриваемом контексте online означает немедленную доступность справочной информации непосредственно из программного обеспечения по сравнению со справочной информацией, доступной в печатной, offline, форме.

Система страниц руководства

Самой известной справочной системой, сопровождающей UNIX практически с момента ее рождения, является справочная система страниц руководства, информация из которой доступна при помощи команд `man`, `apropos` и `whatis`.

Справочная система `man-pages` состоит из отдельных страниц, посвященных отдельным командам, специальным файлам устройств, конфигурационным файлам, системным и библиотечным вызовам и другим понятиям, которые сгруппированы по восьми (обычно, но есть исключения из правил) секциям. Каждая секция имеет заголовочную страницу `intro`, описывающую назначение самой секции (листинг 2.16).

Листинг 2.16 Секции справочной системы man

```
flnn@ubuntu:~$ whatis intro
```

intro (8) - introduction to administration and privileged commands

intro (7) - introduction to overview and miscellany section

intro (3) - introduction to library functions

intro (4) - introduction to special files

intro (1) - introduction to user commands

intro (5) - introduction to file formats and filesystems

intro (6) - introduction to games

intro (2) - introduction to system calls

```
finn@ubuntu:~$ whatis whatis
```

whatis (1) - показывает однострочные описания справочных страниц

```
finn@ubuntu:~$ whatis apropos
```

apropos (1) - поиск в именах справочных страниц и кратких описаниях

```
finn@ubuntu:~$ whatis nan
```

man (1) - доступ к справочным страницам

man (7) - macros to format man pages

Встроенная справка командного интерпретатора

Как было указано ранее, команды интерпретатору могут приводить к вызову внешних программ операционной системы или исполняться непосредственно интерпретатором, будучи встроенными в него.

Внешние программы зачастую документируются отдельными страницами руководства *man* или отдельными справочными страницами *info*, тогда как встроенные команды являются частью интерпретатора и естественным образом описываются все вместе на справочной странице, посвященной интерпретатору (листинг 2.19).

Листинг 2.19 Справка по встроенным командам интерпретатора

```
finn@ubuntu:~$ type cd
```

cd - это встроенная команда bash

```
finn@ubuntu:~$ man cd
```

Нет справочной страницы для cd

```
flnn@ubuntu:~$ whatis cd
```

cd: ничего подходящего не найдено.

```
finn@ubuntu:~$ man bash
```

SHELL BUILTIN COMMANDS

Unless otherwise noted, each builtin command documented in this section as accepting options preceded by - accepts -- to signify the end of the

options. The :, true, false, and test builtins do not accept options and do not treat -- specially. The exit, logout, break, continue, let,

```
cd [-L|[-P [-e]] [-@]][dir]
```

Change the current directory to dir. If dir is not supplied, the value of the HOME shell variable is the default. Any additional ar

Однако обращаться каждый раз к весьма внушительной справке по командному интерпретатору не совсем удобно, поэтому встроенная в командный интерпретатор команда `help` позволяет получить краткую справку по встроенным командам интерпретатора (листинг 2.20).

Листинг 2.20. Встроенная справка командного интерпретатора

```
fint@ubuntu:~$ help -d help
```

```
help - Display information about builtin commands.
```

```
finn@ubuntu:~$ help -d cd
```

```
cd - Change the shell working directory.
```

```
finn@ubuntu:~$ help cd
```

```
cd: cd [-L | [-P [-e]] [-@]] [каталог]
```

```
Change the shell working directory.
```

```
Change the current directory to DIR. The default DIR is the value of the HOME  
shell variable.
```

Пользователи и группы

Как было указано ранее, для начала работы в многопользовательской операционной системе пользователю необходимо «зарегистрироваться», предъявляя имя своей пользовательской учетной записи и пароль, подтверждающий право на ее использование. В результате регистрации в системе запускается командный интерпретатор — первая программа пользовательского сеанса.

Учетные записи (УЗ) служат для авторизации, т. е. для разграничения прав доступа субъектов (процессов пользователей или процессов системных служб) к объектам (файлам, другим процессам, системным вызовам пр.).

Различают пользовательские и групповые учетные записи, при этом каждая пользовательская учетная запись идентифицируется уникальным числовым «пользовательским идентификатором» — UID (User Identifier), а каждая групповая — таким же уникальным числовым «групповым идентификатором» GID (Group Identifier). Именно эти числовые идентификаторы и используются ядром операционной системы при определении и проверке прав доступа субъектов относительно объектов (листинг 2.21).

Учетные записи пользователей используются для аутентификации (проверки подлинности) их при регистрации в системе по имени и паролю, а учетные записи групп — для классификации пользовательских УЗ (по функциям, задачам, ролям, проектам или другими способами) и последующей раздачи прав доступа этим классам» пользователей.

Листинг 2.20 Пользовательские идентификаторы UID и СЮ первичной и дополнительных групп

```
finn@ubuntu:~$ id  
uid=1000(finn) gid=1000(finn) groups=1000(finn),4(adm),..., 130(lxd),131(sambashare)
```

Каждая учетная запись пользователя обязательно связана с одной групповой учетной записью, так называемой «первой» группы пользователя. Исторически сложилось, что в ранней UNIX членство пользователей в группах определялось динамически, т. е. после регистрации пользователя в системе ему (точнее — его процессу) выдавался идентификатор (и, как следствие, права доступа) только одной группы.

Для выполнения действий в другой роли (получения других групповых идентификаторов) нужно было « заново зарегистрироваться в группе » при помощи команды newgrp, предъявляя имя и пароль (!) группы. Позднее (и до сих пор) членство в группе стало статическим, т. е. при регистрации в системе пользователю выдают идентификаторы всех групп, в которых он состоит, при этом первая группа носит название первой (primary), а остальные — дополнительных (supplementary).

Кроме этих основных свойств, каждая учетная запись характеризуется именем домашнего каталога и именем командного интерпретатора (запускаемого при регистрации в системе). Дополнительно, учетная запись пользователя может содержать полное имя пользователя, рабочий и .домашний телефоны, рабочий адрес и прочую информацию, которую можно посмотреть при помощи pinky и finger, а изменить при помощи chfn.

```
finn@ubuntu:~$ finger dvk
```

Учетная запись системного администратора с привилегированными (а точнее неограниченными в буквальном смысле) правами доступа обычно называется root и всегда имеет идентификатор UID=0.

Учетные записи, «от лица которых» выполняются процессы системных служб, называются псевдопользовательскими и идентифицируются в диапазоне UID=1—499 или UID=1—999, а начиная с UID=500 (redhat) или UID=1000 (debian) и далее идентифицируются учетные записи обычных пользователей.

Пользователи и группы, переменные окружения, подсистема управления файлами и вводом-выводом

Лекция 2

Управляющие последовательности

В расширенном режиме видеотерминалов W:[VT52], W:[VT100], W:[VT220] появилась возможность вывода символов в произвольное место экрана и использования полужирного, затемненного, негативного, подчеркнутого и других начертаний символов.

Возможности ввода дополнились функциональными клавишами, клавишами перемещения курсора, дополнительной клавиатуры и пр.

Для этого потребовались дополнительные управляющие символы, которые не поместились в кодировку ascii(7), потому терминалы стали использовать управляющие последовательности символов console_Codes, предваряемые управляющим символом ESC с кодом 6x1B.

Так, например, последовательность ESC # 8 вызовет визуальный тест выравнивания краев терминала, заполнением буквой Е всех строк и столбцов, ESC сбросит терминал в исходное состояние, ESC [In включит полужирное начертание, ESC [2 m – затемненное начертание, ESC [4 m – подчеркивание, а ESC [6 m вернет стандартное начертание символов.

Управляющие символы и их последовательности являются обычными байтами и при литеральном вводе с терминала могут быть сохранены в файл (листинг 2.9), например, при помощи команды tee.

При последующем выводе в на терминал, например при помощи команды cat, будут задействованы соответствующие расширенные возможности. Побайтное содержимое файла можно при этом увидеть на терминале посредством восьмеричного od или шестнадцатеричного дампа hexdump, hd.

Многие «дополнительные» клавиши современных терминалов, такие как функциональные F1...F12, клавиши управления курсором, скроллингом PgUp PgDn и пр., генерируют, управляющие последовательности, которые обрабатываются, например, библиотекой readline и используются для редактирования командной строки.

Несмотря на стандартизацию управляющих последовательностей, разные терминалы все же имеют различия, поэтому в операционной системе появились базы данных с описанием свойств и управляющих последовательностей терминалов terminfo и termcap.

Узнать ESC-последовательности можно при помощи команды infocmp, а вывести их на терминал — включить соответствующий режим — при помощи команды tput.

Примером простейшей программы, использующей управляющие последовательности для форматирования символов при выводе на экран, является утилита просмотра справочных страниц man.

В качестве более изощренных программ, использующих управление расширенными возможностями терминала, можно привести less, nano, mc, многие из которых используют для этого библиотеку ncurses.

А самым экстремальным примером использования управляющих последовательностей терминалов является W:[ASCII-графика] и W: [ANSI-графика], реализующаяся библиотеками aalib и caca, при помощи которых на алфавитно-цифровом (!) терминале можно просматривать видеофильмы (листинг 2.12), например, при помощи видеоплееров mplayer или mpv, поддерживающего эти библиотеки.

Листинг 2.12. Просмотр видео на алфавитно-цифровом терминале

```
finn@ubuntu:~$ setfont Unil-VGA8
```

```
finn@ubuntu:~$ mpv --quiet --vo=caca  
https://www.youtube.com/watch?v=Zo7_00W3GzA
```

```
flnn@ubuntu:~$ youtube-dl --exec 'nplayer -quiet -vo aa:din:bold:reverse'  
https://mmw.youtube.com/watch?v=Zo7_00W3CzA
```

Основной синтаксис командной строки

Основу интерфейса командной строки UNIX составляет командный интерпретатор (КИ), являющийся первой и главной программой, запускаемой в сеансе пользователя.

Двусторонний попеременный диалог с командным интерпретатором начинается с приглашения , в ответ на которое пользователь вводит команду, отправляя ее на выполнение управляющим символом LF 0x0A, получает результат ее исполнения на терминале и новое приглашение, сигнализирующее о готовности КИ к исполнению очередной команды (рис. 2.5).

Многие другие диалоговые программы, например lftp, также будут придерживаться синтаксиса и соглашений, принятых в языке командного интерпретатора.

Базовый синтаксис (подчиняющийся второй части стандарта W:[POSIX]) языка любого командного интерпретатора на самом деле достаточно прост и напоминает язык, близкий к естественному.

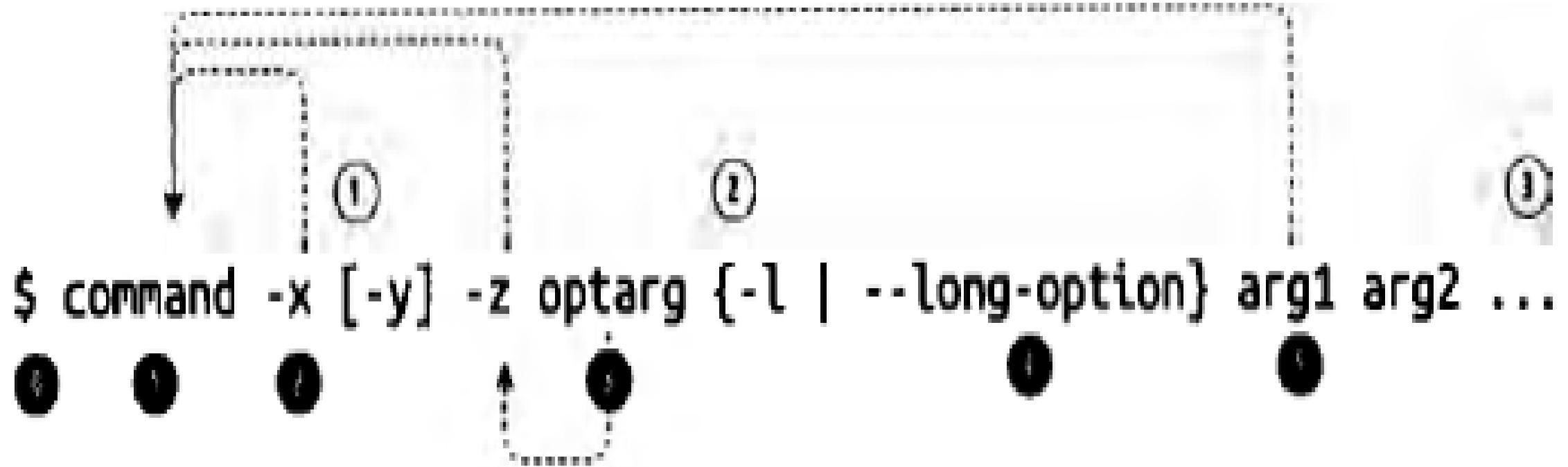


Рис. 2.5. Основной синтаксис командной строки

Например, гп -f -R Изображения Музыка переводится на человеческий как «удалить (rm) без шума и пыли (-f) и со всеми потрохами (-R) каталоги Изображения и Музыка».

Принято говорить, что команда состоит (см. рис. 2.5) из лексем (лексических элементов), разделенных пробельными символами — пробелами SP 0x20 и табуляциями HT 0x09 — в любом количестве и сочетании.

Первая лексема — это название команды, за которой следуют ее параметры: сначала опции (они же — ключи, они же — модификаторы) без аргументов и/или с аргументами, а в конце — аргументы самой команды.

Название команды — это глагол, указывающий, ЧТО делать; опции — это наречия и прочие части речи, объясняющие, КАК это делать; и наконец, аргументы задают то, с ЧЕМ это делать.

В разнообразной документации при описании синтаксиса команды, ее опций и аргументов принято использовать квадратные скобки для указания необязательности опции или аргумента, фигурные скобки и вертикальную черту для указания выбора из вариантов и многоточие для указания повторяемости.

Получив команду, интерпретатор определяет, является ли она псевдонимом, встроенной в интерпретатор, или реализуется внешней программой, подлежащей запуску (листинг 2.13).

Листинг 2.13. утилиты which и type

```
flnn@ubuntu:~$ which date
```

```
/usr/bin/date
```

```
finn@ubuntu:~$ type date
```

```
date является /usr/bin/date
```

```
finn@ubuntu:~$ type -a ls
```

```
ls - это псевдонимом для «ls -*color=auto»
```

```
ls является /usr/bin/ls
```

```
ls является /bin/ls
```

```
finn@ubuntu:~$ type -a pwd
```

```
pwd - это встроенная команда bash
```

```
pwd является /usr/bin/pwd
```

```
pwd является /bin/pwd
```

```
finn@ubuntu:~$ type which
```

```
для which вычислен хэш (/usr/bin/which)
```

```
finn@ubuntu:~$ type type
```

```
type - это встроенная команда bash
```

При наличии нескольких вариантов команды наивысший приоритет имеют псевдонимы, наименьший — внешние команды.

Подстановку псевдонимов можно увидеть, включив трассировку выполнения команд интерпретатора при помощи команды set (листинг 2.14).

Листинг 2.14, ПидЛановка псевдонимов

```
finn@ubuntu:~$ type ls
```

ls - это псевдонимом для «ls --color=auto»

```
finn@ubuntu:~$ set -x
```

```
ftnn@ubuntu:~$ ls -ASs
```

```
+ ls --color=auto -ASs
```

итого 36

```
12 examples.desktop 4 .bashrc 4 .profile 4 .lessht  
4 .cache 4 .bash_history 4 .bash_logout
```

Опции командной строки

В истории развития операционной системы UNIX программы использовали разные способы задания своих опций:

- ◆ односимвольные, например ls -a -l (что эквивалентно ls -l -a или ls -al или ls -la);
- ◆ многосимвольные, например find /var -xdev;
- ◆ длинные, например ps --help;
- ◆ с аргументами, например kill -n 15 1, или kill -nl5 1, или du -B И, или du -BM, или find /etc -type d, или даже ls --sort=size;
- ◆ «нестандартные», например set +x, tar czf tar.tgz ~ или dd if=/dev/dvd of=dvd.iso.

Знак «минус», предваряющий опции, естественно, используется для того, чтобы отличать их от аргументов.

Среди прочих он был выбран потому, что редко встречается как первый символ в аргументах команд (в качестве которых зачастую выступают имена файлов), и еще потому, что на терминале классической UNIX набор более логичного знака «плюс» (что могло бы означать «включить», «активировать») требовал достаточных усилий по нажатию клавиши Shift.

В результате получилось, что, например, в команде set опция x (execution trace) в форме -x включает, а +x выключает трассировку выполнения команд.

В тех редких случаях, когда аргумент команды все же начинается с символа «минус» и тем самым похож на опцию (представим, что нужно выполнить действие над файлом с именем -=filename=-), специальная опция -- сигнализирует о конце списка опций (листинг 2.15), за которым следуют лексемы, обязанные расцениваться как аргументы вне зависимости от их написания.

Листинг 2.15 Конец списка опций

```
finn@ubuntu:~$ stat -=ftlename=-
```

stat: неверный ключ - «=»

По команде «stat --help» можно получить дополнительную информацию.

```
finn@ubuntu:~$ stat -- -=filename=-
```

Файл: -=filename=-

Размер: 0 Блоков: 0 Блок В/В: 4096 пустой обычный файл

Устройство: fc00h/64512d Inode: 26870044 Ссылки: 1

Доступ: (0600/-тм.....)Uid: (1000/ finn) Ctd: (1000/ flnn)

Доступ: 2019-11-17 10:43:36.520984570 +0300 Модифицирован: 2019-11-17
10:43:36.520984570 40300 Изменён: 2019-11-17 10:43:36.520984570 40300

Создан: -

Короткие, односимвольные опции (например, -l -a) без своих аргументов издревле можно было объединять в группы (-la или -al), однако их в этом случае сложно отличать от многосимвольных (-xdev) или односимвольных, склеенных со своими аргументами (-BM).

Поэтому позже появились длинные (в так называемом, GNU- стиле) опции, обозначаемые двумя знаками «минус», позволяющие навести некоторый порядок в виде --block-size=M вместо -BM или, предположим, --dont-descent вместо -xdev.

Справочные системы

Используемые в Linux электронные справочные системы (online help) являются логичным следствием как его родства с семейством операционных систем UNIX — страницы руководства man (manual pages), так и принадлежностью к свободному программному обеспечению под эгидой движения GNU — справочная система info.

Следует отметить, что понятие online в контексте справочных систем вовсе не означает их доступность через Интернет, как это часто, но ошибочно воспринимается сегодня.

В рассматриваемом контексте online означает немедленную доступность справочной информации непосредственно из программного обеспечения по сравнению со справочной информацией, доступной в печатной, offline, форме.

Система страниц руководства

Самой известной справочной системой, сопровождающей UNIX практически с момента ее рождения, является справочная система страниц руководства, информация из которой доступна при помощи команд `man`, `apropos` и `whatis`.

Справочная система `man-pages` состоит из отдельных страниц, посвященных отдельным командам, специальным файлам устройств, конфигурационным файлам, системным и библиотечным вызовам и другим понятиям, которые сгруппированы по восьми (обычно, но есть исключения из правил) секциям. Каждая секция имеет заголовочную страницу `intro`, описывающую назначение самой секции (листинг 2.16).

Листинг 2.16 Секции справочной системы man

```
flnn@ubuntu:~$ whatis intro
```

intro (8) - introduction to administration and privileged commands

intro (7) - introduction to overview and miscellany section

intro (3) - introduction to library functions

intro (4) - introduction to special files

intro (1) - introduction to user commands

intro (5) - introduction to file formats and filesystems

intro (6) - introduction to games

intro (2) - introduction to system calls

```
finn@ubuntu:~$ whatis whatis
```

whatis (1) - показывает однострочные описания справочных страниц

```
finn@ubuntu:~$ whatis apropos
```

apropos (1) - поиск в именах справочных страниц и кратких описаниях

```
finn@ubuntu:~$ whatis nan
```

man (1) - доступ к справочным страницам

man (7) - macros to format man pages

Встроенная справка командного интерпретатора

Как было указано ранее, команды интерпретатору могут приводить к вызову внешних программ операционной системы или исполняться непосредственно интерпретатором, будучи встроенными в него.

Внешние программы зачастую документируются отдельными страницами руководства *man* или отдельными справочными страницами *info*, тогда как встроенные команды являются частью интерпретатора и естественным образом описываются все вместе на справочной странице, посвященной интерпретатору (листинг 2.19).

Листинг 2.19 Справка по встроенным командам интерпретатора

```
finn@ubuntu:~$ type cd
```

cd - это встроенная команда bash

```
finn@ubuntu:~$ man cd
```

Нет справочной страницы для cd

```
flnn@ubuntu:~$ whatis cd
```

cd: ничего подходящего не найдено.

```
finn@ubuntu:~$ man bash
```

SHELL BUILTIN COMMANDS

Unless otherwise noted, each builtin command documented in this section as accepting options preceded by - accepts -- to signify the end of the

options. The :, true, false, and test builtins do not accept options and do not treat -- specially. The exit, logout, break, continue, let,

```
cd [-L|[-P [-e]] [-@]][dir]
```

Change the current directory to dir. If dir is not supplied, the value of the HOME shell variable is the default. Any additional ar

Однако обращаться каждый раз к весьма внушительной справке по командному интерпретатору не совсем удобно, поэтому встроенная в командный интерпретатор команда `help` позволяет получить краткую справку по встроенным командам интерпретатора (листинг 2.20).

Листинг 2.20. Встроенная справка командного интерпретатора

```
fint@ubuntu:~$ help -d help
```

```
help - Display information about builtin commands.
```

```
finn@ubuntu:~$ help -d cd
```

```
cd - Change the shell working directory.
```

```
finn@ubuntu:~$ help cd
```

```
cd: cd [-L | [-P [-e]] [-@]] [каталог]
```

```
Change the shell working directory.
```

```
Change the current directory to DIR. The default DIR is the value of the HOME  
shell variable.
```

Пользователи и группы

Как было указано ранее, для начала работы в многопользовательской операционной системе пользователю необходимо «зарегистрироваться», предъявляя имя своей пользовательской учетной записи и пароль, подтверждающий право на ее использование. В результате регистрации в системе запускается командный интерпретатор — первая программа пользовательского сеанса.

Учетные записи (УЗ) служат для авторизации, т. е. для разграничения прав доступа субъектов (процессов пользователей или процессов системных служб) к объектам (файлам, другим процессам, системным вызовам пр.).

Различают пользовательские и групповые учетные записи, при этом каждая пользовательская учетная запись идентифицируется уникальным числовым «пользовательским идентификатором» — UID (User Identifier), а каждая групповая — таким же уникальным числовым «групповым идентификатором» GID (Group Identifier). Именно эти числовые идентификаторы и используются ядром операционной системы при определении и проверке прав доступа субъектов относительно объектов (листинг 2.21).

Учетные записи пользователей используются для аутентификации (проверки подлинности) их при регистрации в системе по имени и паролю, а учетные записи групп — для классификации пользовательских УЗ (по функциям, задачам, ролям, проектам или другими способами) и последующей раздачи прав доступа этим классам» пользователей.

Листинг 2.21 Пользовательские идентификаторы UID и GID первичной и дополнительных групп

```
finn@ubuntu:~$ id  
uid=1000(finn) gid=1000(finn) groups=1000(finn),4(adm),..., 130(lxd),131(sambashare)
```

Каждая учетная запись пользователя обязательно связана с одной групповой учетной записью, так называемой «первой» группы пользователя. Исторически сложилось, что в ранней UNIX членство пользователей в группах определялось динамически, т. е. после регистрации пользователя в системе ему (точнее — его процессу) выдавался идентификатор (и, как следствие, права доступа) только одной группы.

Для выполнения действий в другой роли (получения других групповых идентификаторов) нужно было « заново зарегистрироваться в группе » при помощи команды newgrp, предъявляя имя и пароль (!) группы. Позднее (и до сих пор) членство в группе стало статическим, т. е. при регистрации в системе пользователю выдают идентификаторы всех групп, в которых он состоит, при этом первая группа носит название первой (primary), а остальные — дополнительных (supplementary).

Кроме этих основных свойств, каждая учетная запись характеризуется именем домашнего каталога и именем командного интерпретатора (запускаемого при регистрации в системе).

Дополнительно, учетная запись пользователя может содержать полное имя пользователя, рабочий и .домашний телефоны, рабочий адрес и прочую информацию, которую можно посмотреть при помощи pinky и finger, а изменить при помощи chfn.

Листинг 2.22. Свойства учетной записи пользователя

```
finn@ubuntu:~$ finger dvk
```

Login: dvk

Name: Dmitry V. Ketov

Directory: /home/dvk

Shell: /bin/bash

Office Phone: +7(812)703-02-02 Home Phone: ---

On since Sun Nov 17 01:51 (MSK) on :0 from :0 (messages off)

On since Sun Nov 17 10:38 (MSK) on pts/1 from 10.0.2.2 5 seconds idle

On since Sat Nov 16 20:59 (MSK) on tty3 34 minutes 19 seconds idle (messages off)

On since Sat Nov 16 21:12 (MSK) on tty4 10 hours 3 minutes idle (messages off)

New mail received Sun Nov 17 11:14 2019 (MSK)

Unread since Sun Nov 16 22:31 2019 (MSK)

No Plan.

Учетная запись системного администратора с привилегированными (а точнее — неограниченными в буквальном смысле) правами доступа обычно называется `root` и всегда имеет идентификатор `UID=0`.

Учетные записи, «от лица которых» выполняются процессы системных служб, называются псевдопользовательскими и идентифицируются в диапазоне `UID=1—499` или `UID=1—999`, а начиная с `UID=500` (`redhat`) или `UID=1000` (`debian`) и далее идентифицируются учетные записи обычных пользователей.

Передача полномочий

Для выполнения определенных административных (привилегированных) действий, например для установки системного времени при помощи команды date, нужны права доступа к определенным системным вызовам.

В классическом UNIX были предусмотрены простые правила разграничения «все или ничего», т. е. все привилегированные действия были разрешены суперпользователю root с UID=0, и никакие привилегированные — всем остальным пользователям.

В этих и подобных ситуациях для администрирования операционной системы непривилегированным пользователям необходимо временно воспользоваться правами суперпользователя, что реализуется посредством классической команды явной передачи полномочий su — switch user, или более поздней команды sudo — switch user do контролируемой передачи полномочий.

Основное различие между su и sudo заключается в том, что команда su реализует «повторную регистрацию в системе», требуя указать имя и ввести пароль того пользователя, чьи полномочия нужно получить.

Напротив, команда sudo реализует явные правила sudoers передачи полномочий, указанные в файле /etc/sudoers, и требует подтвердить передачу полномочий паролем того пользователя, который получает передаваемые полномочия (листинг 2.23).

Листинг 2.23. Передача полномочий.

```
iceking@ubuntu:~$ su -l finn
```

Пароль: <пароль finn'a>

```
finn@ubuntu:~$ id
```

uid=1001(finn) gid=1001(finn) группы=1001(finn)

```
finn@ubuntu:~$ su -1 jake
```

Пароль: <пароль jake'a>

```
jake@ubuntu:~$ id
```

uid=1002(jake) gid=1002(jake) группы=1002(jake)

Продолжение листинга 2.23. Передача полномочий.

```
iceking@ubuntu: ~$ sudo -i -u finn
```

[sudo] пароль для iceking: <пароль iceking'a>

```
finn@ubuntu:~$ id
```

uid=1001(finn) gid=1001(finn) группы=1001(finn)

```
finn@ubuntu:~$ sudo -i -u jake
```

[sudo] пароль для finn: <пароль finn'a>

finn отсутствует в файле sudoers. Данное действие будет занесено в журнал.

Нужно заметить, что в Ubuntu Linux пароль суперпользователя root заблокирован, что не позволяет использовать учетную запись как «обычную» для регистрации в системе и превращает ее в «ролевую».

Как следствие, привилегиями «роли» суперпользователя можно пользоваться лишь при помощи sudo и только непrivилегированным пользователям, явно указанным в правилах передачи sudoers.

Хранилища учетных записей

Информация об идентификаторах UID и GID, именах пользователей и групп, их паролях и прочих свойствах учетных записей размещаются (в простейшем случае) в файловых «хранилищах» — обычных текстовых файлах каталога /etc, формируя базы данных пользовательских /etc/passwd, /etc/shadow и групповых /etc/group, /etc/gshadow учетных записей.

Формат и структура этих файлов хорошо документированы в руководстве passwd, shadow и group, gshadow и представляют собой простейшие текстовые таблицы, где свойства каждой учетной записи представлены набором столбцов одной строки, разделенных символом двоеточия : (листинг 2.24).

Листинг 2.24. Базы данных пользовательских учетных записей

```
finn@ubuntu:~$ cat /etc/passwd
```

```
root:x:0:0:root:/root:/bin/bash
```

```
.....
```

```
dvk:x:1000:1000:Иванов Иван,,+7(812)703-02-02,:/home/dvk:/bin/bash
```

```
.....
```

```
finn@ubuntu:~$ cat /etc/group
```

```
....
```

```
dvk:x:1000:
```

```
lpachin:x:119:dvk,finn
```

```
sambashare:x:131:dvk,finn
```

```
.....
```

При использовании «коммутатора службы имен» (NSS, W:[Name Service Switch]) имеется возможность хранить базы данных учетных записей в любых хранилищах, включая сетевые службы каталогов NIS, NIS+, LDAP, активный каталог Microsoft Windows и даже реляционные сетевые базы данных SQL — при помощи соответствующих модулей NSS (листинг 2.25) и согласно настройкам коммутатора `nsswitch.conf`.

Листинг 2.25. Хранилища пользовательских учетных записей и модули NSS

```
finn@ubuntu:~$ cat /etc/nsswitch.conf
```

```
passwd: files systemd
```

```
group: files systemd
```

```
shadow: files
```

```
finn@ubuntu:~$ find /lib/ -name 'libnss'
```

```
/lib/x86_64-linux-gnu/libnss_systemd.so.2
```

```
/lib/x86_64-linux-gnu/libnss_files.so
```

```
/lib/x86_64-linux-gnu/libnss_winbind.so.2
```

```
finn@ubuntu:~$ dpkg -S /lib/x86_64-linux-gnuAibnss_winbind.so.2
```

```
libnss-winbind:amd64: /lib/x86_64-linux-gnu/libnss_winbind.so.2
```

```
finn@ubuntu:~$ dpkg -s libnss-winbind Package: libnss-winbind
```

This package provides `nssjwinbind`, a plugin that integrates with a local `winbindd` server to provide user/group name lookups to the system; and `nssjwins`, which provides hostname lookups via both the NBNS and NetBIOS broadcast protocols.

Переменные окружения и конфигурационные dot-файлы

Для одноразовой параметризации выполнения команд служат их индивидуальные ключи, указываемые каждый раз при запуске команды, но иногда требуется установить некий параметр, который бы действовал в течение всего сеанса работы пользователя с системой, или общий параметр, который действовал бы для всех команд, запускаемых в сеансе.

Таким механизмом являются окружение `environ` и переменные окружения, значения которых можно увидеть при помощи команды `env`.

Переменные окружения обычно документируются на странице руководства к тем программам, на которые воздействуют, как правило, в разделе `ENVIRONMENT`.

Например, переменная окружения `PATH` содержит перечисление разделенных символом `:` имен каталогов, где любой командный интерпретатор ищет одноименные запускаемым командам программы (листинг 2.26).

Листинг 2.26. Переменная окружения PATH

```
ftnn@ubuntu:~$ date
```

```
Вс ноя 17 11:31:24 MSK 2019
```

```
finn@ubuntu:~$ help -d unset
```

```
unset - Unset values and attributes of shell variables and functions.
```

```
flnn@ubuntu:~$ unset PATH
```

```
finn@ubuntu:~$ date
```

```
bash: date: Нет такого файла или каталога
```

Переменные окружения `LANGUAGE` и `LANG` содержат идентификаторы языка, на котором программы стараются общаться с пользователем (листинг 2.27); например, `man` ищет перевод страницы руководства.

Если точнее, то переменная `LANGUAGE` на самом деле определяет список языков, в порядке которого определяется язык общения, т. к. далеко не все программы имеют переводы, а если и имеют, то не на все языки.

Переменная `LANG` определяет язык по умолчанию, если в порядке просмотра `LANGUAGE` ничего подходящего не найдено.

Набор переменных окружения `LC_*` определяет другие языковые особенности, отличные от языка сообщений; например, переменная `LC_TIME` определяет формат вывода даты и времени.

Кроме того, переводы устанавливаются в систему из специальных языковых пакетов, а список доступных можно увидеть при помощи команды `locale`.

Листинг 2.27. Переменные окружения LANGUAGE LANG и 1C_*

```
flnn@ubuntu:~$ date  
Вс ноя 17 12:55:36 MSK 2019  
flnn@ubuntu:~$ locale  
LC_TIME=ru_RU.UTF-8  
flnn@ubuntu:~$ locale -a  
C  
C.UTF-8  
POSIX  
en_CB.utf8  
en_US.utf8  
ko_KR.utf8  
ru_RU.utf8  
ru_UA.utf8  
finn@ubuntu:~$ export LC_TIME=ko_KR.UTF-8  
finn@ubuntu:~$ date  
2019. 11. 17. 13:33:12 MSK  
finn@ubuntu:~$ cal
```

11월 2019						
일	월	화	수	목	금	토
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

Продолжение листинга 2.27

```
finn@ubuntu:~$ tar
```

tar: Необходимо указать один из параметров «-Acdtrux», «--delete» или «--test-label»

Попробуйте «tar --help» или «tar --usage» для получения более подробного описания.

```
finn@ubuntu:~$ locale
```

LANGUAGE=ru:en

```
ftnn@ubuntu:~$ export LANGUAGE=ko:ru:en
```

```
finn@ubuntu:~$ tar
```

tar: '-Acdtrux', '--delete', '--test-label' <корейские иероглифы>

```
flnn@ubuntu:~$ зфыыwhatis passwd
```

passwd (5) –

passwd (1) - изменяет пароль пользователя

passwd (1ssl) - compute password hashes

Переменная окружения PAGER указывает имя программы «листателя», использующегося другими программами, вывод которых не умещается на один экран.

Так, например, поступает `tail` при отображении отформатированной страницы руководства, `mail` — при просмотре длинного письма, `mysql` или `psql`- при выводе большого количества результирующих строк ответа за запрос к базе данных.

Наиболее распространенным «листателем» является `less`, используемый как замена менее удобного «классического» `more`.

Переменные окружения EDITOR и VISUAL указывают имя текстового редактора, который будет вызван другими программами при необходимости редактировать текст.

Например, `crontab` использует указанный таким образом редактор для изменения списка периодических заданий, `mail` — для редактирования отправляемого сообщения, `mysql` или `psql` - для редактирования длинных запросов к базе данных, `lftp` - для редактирования списка «закладок».

Очень часто в качестве редактора используется «классический» и достаточно непривычный `vi`, который можно таким образом заменить более удобным `nano`.

Переменная BROWSER указывает имя просмотрщика HTML (листинг 2.28), который будет использован другими программами при необходимости показать HTML-страницу, например отформатированную таким образом страницу руководства man.

Листинг 2.28 Переменная окружения BROWSER

```
finn@ubuntu:~$ man -H ls
```

```
sh: 1: exec: www-browser: not found
```

```
man: couldn't execute any browser from exec www-browser
```

```
ftnn@ubuntu:~$ export BROWSER=chromium-browser
```

```
finn@ubuntu:~$ man -H ls
```

В текущем сеансе браузера создано новое окно.

Некоторые программы имеют специальную переменную окружения, которая содержит ключи, применяемые каждый раз при вызове программы, например MANOPT для tar. Такие переменные для программ XXX чаще всего имеют имя XXXOPT или XXX.OPTIONS или даже XXX, например ZIPOPT для zip, TAR_OPTIONS для tar, gzip для gzip и LESS/MORE для less и тоге соответственно.

Установив, например, MANOPT=-H (BROWSER=chromium-browser, или BROWSER=firefox, или BROWSER=links, или BROWSER=lynx), можно просматривать страницы руководства в (одном из указанных графическом и/или текстовом) Web-браузере, а установив GZIP=-9, можно заставить упаковщик всегда использовать девятую (самую сильную, но самую медленную) степень сжатия.

Переменная окружения PS1 изменяет приглашение командного интерпретатора и может содержать подстановки (документированные в bash, см. раздел PROMPTING), например, \u — имя зарегистрированного в системе пользователя, \h — короткое собственное имя компьютера, \w — имя текущего каталога и \\$ символ приглашения \$ для обычного пользователя и # для суперпользователя root, \t — время в 24-часовом формате, \e — управляющий символ ESC и пр.

Используя подстановки PS1 и управляющие ESC-последовательности console_codes или воспользовавшись базой данных управляющих ESC-последовательностей termininfo и командой tput, можно модифицировать приглашение по своему предпочтению (листинг 2.29).

Листинг 2.29. Переменная окружения PS1

```
finn@ubuntu:~$ man 5 terminfo  
flnn@ubuntu:~$ tput rev | od -a  
finn@ubuntu:~$ tput din | od -a  
finn@ubuntu:~$ tput sgr0 | od -a  
finn@ubuntu:~$ PS1='\e[2n\t\n\e(B\ue[n\u@\h \e[7n\w\ue(B\ue[n \$ '
```

Или даже можно воспользоваться поддержкой символов unicode в UTF-8 представлении на терминале1 (листинг 2.30).

В этом примере эмулятор терминала в графическом интерфейсе по умолчанию имеет нужные шрифты, а на консоли необходимо загрузить подходящий Unicode-шрифт, например командой setfont Uni2-Terninus16.

Листинг 2.30 Переменная окружения PS1

```
finn@ubuntu:~$ stty  
speed 38400 baud; line = 0;  
eol = M-^?; eol2 = M-^?; swtch = M-^?;  
ixany iutf8  
finn@ubuntu:~$ PS1='\u\342\230\273 \h:\w\$ '
```

Переменная окружения TERM устанавливает имя терминала, по которому программы, использующие управляющие ESC-последовательности (например, файловый менеджер mc), берут их значения из базы данных terminfo.

Для эмуляторов терминала в графическом интерфейсе ее значение обычно TERM=xterm или TERM=xterm-color, для консоли Linux TERM=linux, а для настоящего аппаратного терминала W:[VT100] TERM=vt100.

При неправильном значении переменной (листинг 2.31), например, могут «перестать работать» функциональные клавиши просто потому, что программа будет ожидать поступление определенной ESC-последовательности, соответствующей функциональной клавише, а в реальности будет поступать другая.

Листинг 2.31. Переменная окружения TERM

```
finn@ubuntu:~$ env  
TERM=xterm  
finn®ubuntu:~$ TERM=linux
```

```
finn@ubuntu:~$ infocmp  
finn@ubuntu:~$ od -a
```

Стоит отметить, что переменные окружения сохраняют свои установленные значения в оперативной памяти командного интерпретатора и теряют их при завершении сеанса.

Для установки постоянных значений параметров программ логично поместить их в какое-либо долгосрочное хранилище, например в файлы на диске.

Специальные файлы и каталоги, сохраняющие конфигурационные параметры, по соглашению имеют имена, начинающиеся с точки (англ. dot — точка), располагаются в домашнем каталоге пользователя и носят название dot-файлов (листинг 2.32).

Листинг 2.32. Конфигурационные dot-файлы

```
finn@ubuntu:~$ ls -A
```

```
.bash_history
```

```
.bash_logout
```

```
.bashrc
```

```
.profile
```

```
.lftp
```

```
.ssh
```

```
finn@ubuntu:~$ file .profile .bashrc .lftp .ssh
```

```
.profile: ASCII English text .bashrc: ASCII English text .lftp: directory
```

```
.ssh: directory
```

Некоторые программы, например lftp, ssh или ssh, имеют собственные конфигурационные файлы и/или каталоги, тогда как простейшие less, tar и zip предполагают, что «постоянные» параметры по-прежнему задаются при помощи переменных окружения LESS, TAR_OPTIONS и ZIPOPT.

В таких случаях <постоянныe> значения переменных сохраняются в каком-либо конфигурационном файле командного интерпретатора bash, например считываемом в начале сеанса — .profile или при каждом запуске — .bashrc (листинг 2.33).

Листинг 2.32. Конфигурационные dot-файлы интерпретатора bash

```
finn@ubuntu:~$ less -./.bashrc
```

```
GZIP=-9v
```

```
PS1='\[2m\t\[e(B\[e[m \u@\h \e[7m\w\[e(B\[e[m \$ '
```

Конфигурационные dot-файлы представляют собой текст на некотором языке, понятном конфигурируемой программе, например язык командного интерпретатора bash используется в .profile и .bashrc.

В большинстве случаев имена dot-файлов и их язык документируются в страницах руководства к «их» программам, обычно в разделе FILES.

Нередко конфигурационному файлу посвящается отдельная страница в пятой секции, например паногс для dot-файла .nanogс текстового редактора nano или netrc для файла .netrc FTP-клиентов ftp и lftp, или ssh_config для .ssh/config SSH-клиента ssh.

В отдельных случаях, когда конфигурируется не конкретная программа, а общая для многих программ библиотека, например readline, название переменных окружения, имя и язык конфигурационного файла можно получить (INPUTRC и .inputrc для readline, соответственно) из страницы руководства самой библиотеки/

readline — библиотека расширенного редактирования вводимой командной строки, которая используются bash, lftp), mysql и пр.

Командный интерфейс Linux, каким бы «страшным» ни казался, в реальности удивительно функционален для решения массы разнообразных задач, хотя он и не решает абсолютно все задачи одинаково эффективно.

Например, его невероятно сложно и неудобно использовать для обработки графической информации, когда при взаимодействии с пользователем требуется ввести колоссальное количество «графических» данных, например указать обтравочный контур. В этом случае графический интерфейс с «непосредственным» манипулированием подойдет гораздо лучше.

Для начинающего пользователя интерфейс командной строки действительно является непривычным, что зачастую путают с неудобством, так толком и не разобравшись со всеми его возможностями. Вся сила языка командного интерпретатора в полной мере раскрывается в дальнейшем материале, до освоения которого предлагается не делать скоропалительных выводов.

Естественные языки, которые используют люди для взаимодействия между собой, на порядок сложнее формального командного языка операционной системы.

Однако использование глаголов (команд), существительных (аргументов) и наречий (опций) родного языка мало у кого вызывает чувство неудобства.

Наоборот, странным покажется тот человеческий индивидуум, который попытается в обществе использовать непосредственное манипулирование, например указывая (щелкая) пальцем в магазине на товары (значки) и мыча что-то нечленораздельное.

Скорее всего, мы примем его за иностранца (или это будет ребенок), еще не в полной мере владеющего языком.

Именно командный интерфейс в современном виде — аудиоформе — больше не является уделом художественных фантастических произведений, где капитаны межгалактических кораблей командуют кораблям «включить защитное поле»

Теперь мы все можем при помощи командного аудиоинтерфейса смартфона найти ближайшую пицерию или маршрут к нужному месту.

Подсистема управления файлами и вводом-выводом

Все операционные системы семейства W:[UNIX], включая Linux, базируются на одной универсальной идее, заложенной в их общем предке, определившем основные черты семейства — операционной системе UNICS.

В аббревиатуре UNICS, или же UNiplexed Information & Computing Service, центральное место занимает идея «uniplex»ирования, или же односоставности (односложности) — идея решать разные задачи единым способом.

Одним из выражений этой идеи является утверждение о том, что информация есть файл, откуда бы эта информация в систему ни поступала.

При помощи файлов обеспечивается доступ к информации на устройствах хранения (записанной ранее), информации с устройств связи (принимаемой из каналов связи в реальном времени), информации из любых других источников.

Файл, таким образом, является единицей обеспечения доступа к информации, а не единицей ее хранения, как в других операционных системах.

Одни файлы обеспечивают доступ к информации, хранимой на разнообразных носителях: магнитных дисках и дискетах, оптических CD/DVD/BD, твердотельных «дисках» и пр.

Другие файлы обеспечивают доступ к информации, поступающей из/в устройств ввода-вывода — клавиатур, манипуляторов «мышь», тачпадов, сенсорных экранов, последовательных и параллельных портов, видеокамер, звуковых карт и пр.

Особенные файлы обеспечивают доступ к информации о сущностях ядра операционной системы (процессы, нити, модули, драйвера и пр.).

Так или иначе, все файлы одинаково идентифицируются своими именами, упорядоченными в форме единой и единственной иерархической структуры, называемой деревом каталогов (рис. 3.1).

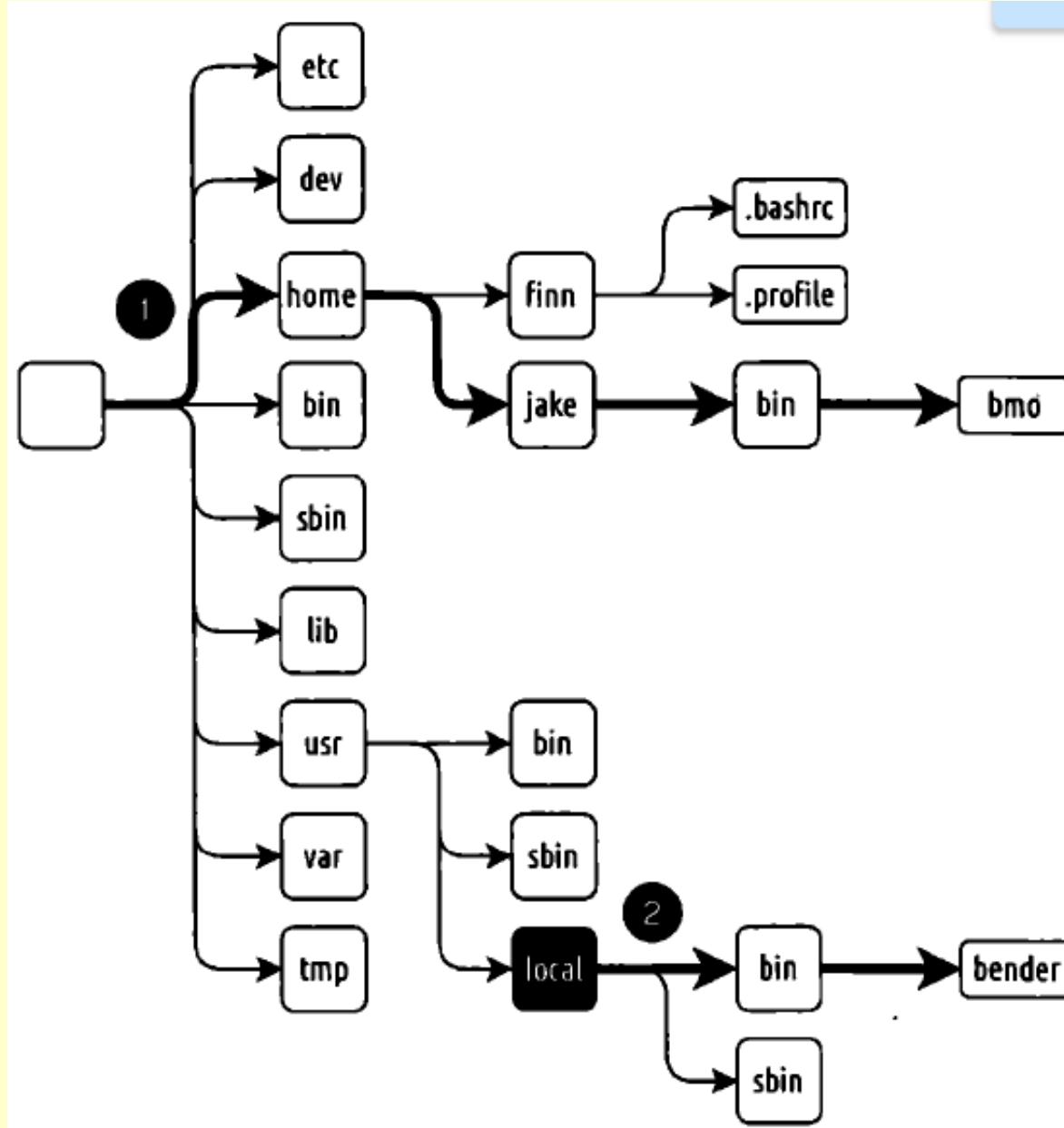


Рис. 3.1. Абсолютные и относительные путевые имена

Путевые имена файлов

Глобально уникальным идентификатором файла в пределах операционной системы является его абсолютное путевое имя, определяемое как путь от корня дерева каталогов до целевого файла, включая начало и конец пути.

Необходимо акцентировать внимание на том, что имя у корневого каталога отсутствует, т. е. является пустой строкой. Таким образом, абсолютное путевое имя файла *bmo* (см. рис. 3.1) записывается как разделенные символом / имена всех каталогов пути и имя самого файла, включая концы – */home/jake/bin/bmo*.

Относительное путевое имя вычисляется как остаток пути от некоторого заранее заданного (называемого рабочим, WD — working directory) каталога, до целевого файла, включая конец пути. Для рабочего каталога WD: */usr/local* относительное путевое имя файла *bender* записывается как разделенные символом / имена всех каталогов остатка пути и имя самого файла – */bin/bender*.

Некоторые каталоги дерева (например, каталоги первого уровня) носят устоявшиеся в семействе операционных систем UNIX имена) и дифференцируют содержание по смысловому признаку.

Например, каталог `/bin` (`binary`) предназначен для системных программ общего назначения, каталог `/usr/bin` – для прикладных (условно) программ общего назначения, каталог `/usr/local/bin` – для «локально» установленных прикладных программ общего назначения, а каталоги `bin` внутри домашних каталогов пользователей – для программ персонального назначения.

Аналогично определяется назначение каталогов `/sbin`, `/usr/sbin`, `/usr/local/sbin` с той лишь разницей, что каталоги `sbin` расшифровываются как `superuser's binaries` и предназначаются для программ системного администрирования: системных, прикладных и «локально установленных» соответственно. Каталоги `/lib`, `/usr/lib` и `/usr/local/lib` аналогично содержат системные и прикладные библиотеки.

Каталог `/etc` содержит общесистемные конфигурационные файлы и с полным правом может расшифровываться как `editable text configuration`.

Каталог `/home` является контейнером домашних каталогов пользователей (кроме суперпользователя `root`).

Каталог `/var` служит хранилищем динамических данных (журнальные файлы `/var/log`, почтовые ящики `/var/mail`, разнообразные очереди `/var/spool` и подобное), а каталог `/tmp` выступает хранилищем временных данных.

Каталоги `/dev`, `/proc` и `/sys` содержат специальные файлы устройств и файлы псевдофайловых систем `proc` и `sysfs`.

Типы файлов

Файлы как единицы обеспечения доступа к данным различаются операционной системой по типам, указывающим источник информации. Обычные (*regular*) файлы и каталоги (*directory*) обеспечивают сохранение информации на тех или иных носителях.

Специальные файлы устройств (*special device file*) позволяют обмениваться информацией с тем или иным аппаратным устройством ввода-вывода, а именованные каналы и файловые сокеты предназначены для обмена информацией между процессом одной программы и процессами других программ.

В примере из листинга 3.1 в полном (-l, *long*) выводе команды `ls` проиллюстрирован признак типа файла.

Символом - обозначается обычный файл, символом b или c — специальные файлы блочного (*block*) или символьного (*character*) устройства, символом p — именованный канал (*pipe*), символом s — сокет (*socket*), а символом l — символьическая ссылка (*link*).

Листинг 3.1. Признак типа файлов ls

```
finn@ubuntu:~$ ls -l /bin/ls /dev/sda /dev/tty /sbin/halt  
-rwxr-xr-x 1 root root 142144 сен 5 13:38 /bin/ls  
brw-rw--- 1 root disk 8, 0 ноя 17 03:31 /dev/sda  
crw-rw-rw- 1 root tty 5, 0 ноя 17 12:18 /dev/tty  
lrwxrwxrwx 1 root root 14 ноя 13 00:20 /sbin/halt -> /bin/systemctl  
finn@ubuntu:~$ ls -l /run/initctl /run/udev/control  
prw..... 1 root root 0 ноя 17 03:30 /run/initctl  
srw..... 1 root root 0 ноя 17 03:30 /run/udev/control
```

Обычные файлы

Обычные файлы содержат пользовательскую информацию: текст, изображения, звук, видео и прочие данные в виде набора байтов (см. на рис. 3.2, листинг 3.2).

За структуру содержания и имена обычных файлов ответственны прикладные программы, а операционная система не накладывает никаких ограничений

Листинг 3.2 Содержание обычных файлов

```
finn@ubuntu:~$ file /usr/share/man/man1/file.l.gz
```

```
/usr/share/man/man1/file.1.gz: gzip compressed data, max compression, from Unix,  
original size modulo 2A32 21484
```

```
finn@ubuntu:~$ file /etc/passwd
```

```
/etc/passwd: ASCII text
```

```
finn@ubuntu:~$ file /bin/ls
```

```
/bin/ls: ELF 64-bit LSB pie executable, X86-64, version 1 (SYSV), dynamically linked,  
interpreter /lib64/ld-linux-x86-64.so.2,
```

```
BuildID[sha1]=2f15ad836be3339dec0e2e6a3c637e08e48aacbd, for GNU/Linux 3.2.0,  
stripped
```

```
finn@ubuntu:~$ file /usr/share/sounds/alsa/Noise.wav
```

```
/usr/share/sounds/alsa/Noise.wav: RIFF (little-endian) data, WAVE audio, Microsoft PCM,  
16 bit, mono 48000 Hz
```

```
finn@ubuntu:~$ file /usr/share/backgrounds/Sky_Sparkles_by_Joe_Thompson.jpg
```

```
/usr/share/backgrounds/Sky_Sparkles_by_Joe_Thompson.jpg: JPEG image data, JFIF  
standard 1.01, aspect ratio, density 1x1, segment length 16, progressive, precision 8,  
3840x2160, components 3
```

Создать обычный файл можно при помощи любой программы, сохраняющей информацию в файл, например посредством текстовых редакторов `vi`, `nano` или `mcedit`.

Для создания пустого обычного файла можно воспользоваться командой `touch` — см. листинг 3.5.

Для удаления обычного файла предназначается команда `rm` — См. листинг 3.6.

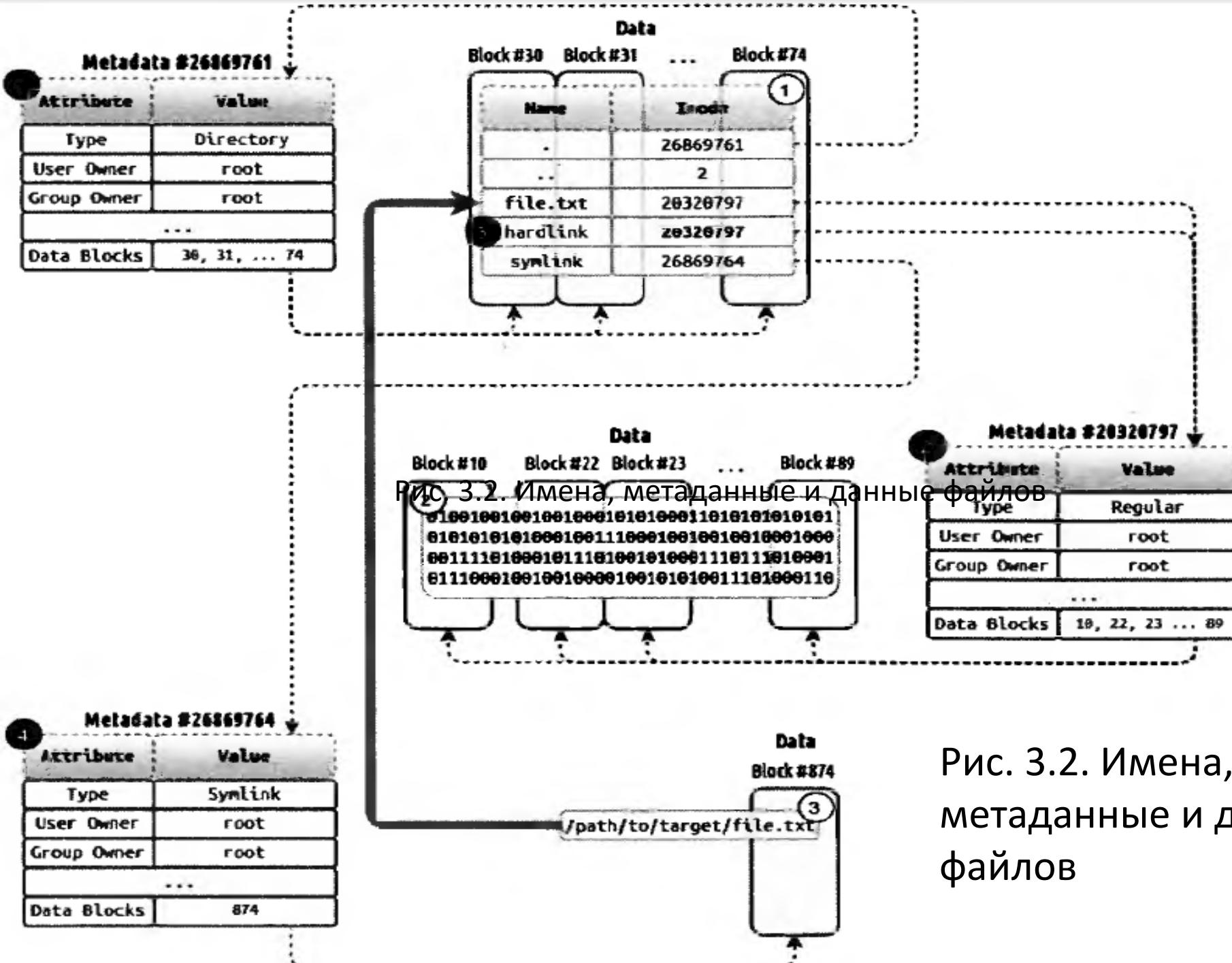


Рис. 3.2. Имена, метаданные и данные файлов

Каталоги

Файлы-каталоги, в отличие от обычных файлов, имеют служебное для операционной системы содержимое — таблицу имен (см. на рис. 3.2) файлов и соответствующих им номеров индексных дескрипторов (inode, index node), проиллюстрированных в листинге 3.3.

Листинг 3.3. Имена и номера индексных дескрипторов файлов finn@ubuntu:~\$ ls -at

```
20332580 . 20318930 .bash_logout 20320866 examples.desktop
```

```
20316161 .. 20320868 .bashrc 20320867 .profile
```

```
20320797 .bash_history 20332712 .cache
```

Каждый индексный дескриптор содержит метаданные (см. на рис. 3.2) — список стандартных свойств файла, в том числе указывающих местоположение данных файла (набора блоков) на файловой системе.

Полный набор метаданных (листинг 3.4) позволяет получить команда stat, включая размер файла , количество занимаемых блоков на диске , тип файла , номер индексного дескриптора , права доступа , владельцев и пр.

Листинг 3.4. Метаданные файлов

```
finn@ubuntu:~$ stat .profile
```

Файл: .profile

Размер: 807 Блоков: 8 Блок В/В: 4096 обычный файл

Устройство: 802h/2050d Inode: 393433 Ссылки: 1

Доступ (0644/-rw-r--r--)Uid: (1001/ fin.y Gid: (1001/ firm)

Доступ: 2019-11-17 15:14:31.673047212 +0300

Модифицирован: 2019-11-13 00:25:26.723714502 +0300

Изменён: 2019-11-13 00:25:26.723714502 +0300

Создан: -

Для создания каталогов предназначена команда `mkdir`, а для удаления — `rmdir`, при этом удалению подлежат только пустые каталоги (см. листинг 3.8).

Имена, данные, метаданные и индексные дескрипторы

Каждый раз, когда используется путевое (абсолютное или относительное) имя файла, производится итеративный поиск файла в дереве путем последовательного разбора на имя (по которому можно найти метаданные и данные) первого каталога пути, содержащего, в свою очередь, имя второго каталога пути, который содержит имя третьего и т. д., пока в конце поиска не будут найдены имя, метаданные и данные указанного файла.

Такая ссылочность позволяет сформировать удобную древовидную структуру для каталогизации файлов (называемую деревом каталогов), однако сам поиск является относительно длительной операцией.

Ссылки

Листинг 3.5 Жесткая ссылка

```
finn@ubuntu:~$ touch readne
```

```
finn@ubuntu:~$ ls -li readme
```

```
20318653 -rw-r--r-- 1 finn finn 0 апр. 1 01:22 readme
```

```
finn@ubuntu:~$ ln readne readne.txt
```

```
finn@ubuntu:~$ touch README
```

```
flnn@ubuntu:~$ ls -li readne readne.txt README
```

```
20318653 -rw-r--r-- 2 finn finn 0 апр. 1 01:22 readne
```

```
20319121 -rw-r--r-- 1 finn finn 0 апр. 1 01:23 README
```

```
20318653 -rw-r--r-- 2 finn finn 0 апр. 1 01:22 readne.txt
```

Более того, оба имени являются равнозначными, и нет возможности узнать, какое из них создано первым, из чего нужно заключить, что первое и единственное имя файла уже является его жесткой ссылкой (на номер индексного дескриптора).

При добавлении файлу нового имени (жесткой ссылки) в его метаданных увеличивается счетчик количества имен (см. в листинге 3.6), а при удалении файла сначала удаляется имя и уменьшается счетчик количества имен , и только при удалении последнего имени высвобождаются метаданные и данные файла.

Листинг 3.6 Счетчик файлов

```
finn@ubuntu:~$ ln readme read.me
```

```
finn@ubuntu:~$ ls -li read*
```

```
20318653 -rw-r--r-- 0 3 finn finn 0 anp.1 01:22 readme
```

```
20318653 -rw-r--r-- 3 finn finn 0 anp.1 01:22 read.me
```

```
20318653: -rw-r--r-3 finn finn 0 anp.1 01:22 readme.txt
```

```
finn@ubuntu:~$ m readme
```

```
finn@ubuntu:~$ ls -li read*
```

```
20318653 -rw-r--r-- ② 2 finn finn 0 anp.1 01:22 read.me
```

```
20318653 -rw-r-r-2 finn finn 0 anp.1 01:22 readme.txt
```

```
finn@ubuntu:~$ rn readne.txt finn@ubuntu:~$ ls -li read*
```

```
20318653 -rw-r--r-- ② 1 finn finn 0 anp.1 01:22 read.me
```

Нужно заметить, что удаление файла — двухшаговая операция, состоящая из удаления имени файла, а затем — удаления метаданных (и высвобождения блоков, занимавшихся этим файлом).

Удаление метаданных файла не выполняется вообще, если у файла еще остались имена (жесткие ссылки), и не происходит сразу, если файл открыт каким-либо процессом.

Метаданные и блоки, занимаемые файлом, высвобождаются только при закрытии этого файла всеми открывшими его процессами, что проиллюстрировано в примере из листинга 3.7.

Команда `df`(`D` измеряет доступное (свободное, `disk free`) место на файловой системе указанного файла, тогда как команда `du`, наоборот, измеряет занимаемое (`disk usage`) указанным файлом место на его файловой системе.

Листинг 3.7. Удаление открытого файла

```
finn@ubuntu:~$ df -h .
```

Файл.система Размер Использовано Дост Использовано% Смонтировано в
/dev/mapper/ubuntu - root 455G (400G 32G** 93% /

```
finn@ubuntu:~$ du -sh astra-linux-l.3-special-edition-smolensk-disk3-devel.iso  
2,8G astra-linux-l.3-special-edition-snolensk-disk3-devel.iso
```

```
finn@ubuntu:~$ m astra-linux-l.3-special-edition-snolensk-disk3-devel.iso
```

```
finn@ubuntu:~$ df -h .
```

Файл.система Размер Использовано Дост Использовано% Смонтировано в
? /dev/mapper/ubuntu-root 455G 400G 32G*» 93% /

```
finn@ubuntu:~$ lsof astra-linux-l.3-special-edition-snolensk-disk3-devel.iso COMMAND PID USER FD TYPE  
DEVICE SIZE/OFF NODE NAME
```

```
fuseiso 16925 finn 3r REG 252,0 2947385344 20316584 astra-linux-1.3-special-edition-  
Smolensk-disk3-devel.iso
```

```
finn@ubuntu:~$ kill 16925
```

```
finn@ubuntu:~$ df -h .
```

Файл.система Размер Использовано Дост Использовано% Смонтировано в
! /dev/mapper/ubuntu-root 455G 397G 35G*» 92% /

Специальные имена текущего и родительского каталогов на поверху тоже оказываются жесткими ссылками, поэтому у любого каталога по крайней мере два имени — свое «собственное» в родительском и специальное . в самом себе, а у каталогов с подкаталогами еще и имена в каждом из дочерних (листинг 3.8).

Листинг 3.8. Имена каталогов

```
finn@ubuntu:~$ mkdir folder
```

```
finn@ubuntu:~$ ls -ldt folder
```

```
20357139 drwxr-xr-x. 2 finn finn 4096 апр.
```

```
finn@ubuntu:~$ cd folder
```

```
finn@ubuntu:~/folder$ ls -lai итого 8
```

```
20357139 drwxr-xr-x 2 finn finn 4096 апр.
```

```
20332580 drwxr-xr-x 4 finn finn 4096 апр.
```

```
inn@ubuntu:~/folder$ mkdir child
```

```
finn@ubuntu:~/folder$ cd child
```

```
finn@ubuntu:~/folder/child$ ls -lai
```

```
итого 8
```

Существенным ограничением жесткой ссылки в дереве каталогов, куда смонтирована более чем одна файловая система, является локальность жесткой ссылки в пределах своей файловой системы в силу локальной значимости номеров индексных дескрипторов.

Так как на каждой новой файловой системе номера индексных дескрипторов начинают нумероваться с нуля, то жесткая ссылка всегда указывает на метаданные файла в «своей» файловой системе и не может указывать на метаданные файла в «чужой» файловой системе общего дерева каталогов.

Для преодоления этого ограничения служит символьическая ссылка `symlink`, являющаяся самостоятельным служебным типом (см. на рис. 3.2) и содержащая путевое имя (см. на рис. 3.2 и в листинге 3.9) к целевому файлу.

Листинг 3.9. Символическая ссылка

```
finn@ubuntu:~$ ln -s read.me readne.lst
```

```
finn@ubuntu:~$ ls -li read*
```

```
20318653 -rw-r--r-- 1 finn finn 0 апр. 1 01:22 read.me
```

```
20319944 lrwxrwxrwx 1 finn finn 7 апр. 2 00:03 readne.lst -> read.me
```

В случае с символической ссылкой при удалении целевого файла сама ссылка будет указывать в никуда и называться «сиротой» (orphan). Попытка прочитать (листинг 3.10) такую ссылку приводит к странным на первый взгляд результатам: файл «существует» для команды ls, но команда просмотра содержимого cat говорит об обратном.

Ничего удивительного, если помнить, что ls работает с именами файлов, а cat — с их данными (которые действительно не существуют).

Листинг 3.10 Сиротская ссылка

```
finn@ubuntu:~$ гп read.me
```

```
finn@ubuntu:~$ ls read*
```

```
readme.1st
```

```
ftnn@ubuntu:~$ cat readme.1st
```

```
cat: readme.1st: Нет такого файла или каталога
```

```
finn@ubuntu:~$ ls -l read*
```

```
lrwxrwxrwx 1 finn finn 7 апр. 2 00:03 readme.1st -> read.me
```

```
finn@ubuntu:~$ cat read.me
```

```
cat: read.me: Нет такого файла или каталога
```

Символические ссылки могут ссылаться на имена друг друга неограниченное количество раз, а при попытке использовать одну из таких ссылок будут прочитаны данные файла, последнего в цепочке ссылок.

По ошибке можно даже закольцевать (листинг 3.11) две или более ссылок, с чем разберется операционная система при чтении одной из ссылок.

Листинг 3.11 Кольцевые ссылки

```
finn@ubuntu:~$ ln -s readme.1st read.me
finn@ubuntu:~$ ls -l read*
lrwxrwxrwx 1 finn finn 10 апр. 2 00:41 read.ne -> readme.1st
lrwxrwxrwx 1 finn finn 7 апр. 2 00:03 readme.1st -> read.ne
finn@ubuntu:~$ cat read.me
cat: read.me: Слишком много уровней символьных ссылок
```

Основным назначением символьических (и изначально, жестких) ссылок является «множественная каталогизация» файлов, т. е. различные наборы разных имен одних и тех же данных.

Типичным примером использования ссылок является организация boot сценариев запуска системных служб при старте операционной системы.

Сами сценарии располагаются в каталоге `/etc/inlt.d`, а в каталогах `/etc/rcS.d`, `/etc/rc0.d`, ..., `/etc/rc6.d` размещены символические ссылки на эти сценарии, которые должны быть запущены с определенными параметрами при переключении состояния системы между так называемыми «уровнями исполнения» (листинг 3.12).

Листинг 3.12 Сценарий запуска служб и их каталогизация по уровням исполнения

```
flnn@ubuntu:~$ ls -l /etc/rc?.d
```

```
/etc/rc2.d:
```

```
lrwxrwxrwx 1 root root 20 ноя 13 00:16 /etc/rc2.d/S19postgresql -> ../init.d/postgresql
```

```
lrwxrwxrwx 1 root root 17 ноя 13 00:16 /etc/rc2.d/S20postfix -> ../init.d/postfix
```

```
/etc/rc6.d:
```

```
lrwxrwxrwx 1 root root 17 ноя 13 00:16 /etc/rc6.d/K20postfix -> ../init.d/postfix
```

```
lrwxrwxrwx 1 root root 17 ноя 13 00:16 /etc/rc6.d/K21postgresql -> ../init.d/postgresql
```

В этом примере сценарии postfix и postgresql имеют вторичные имена, начинающиеся с K в каталоге rc6.d, и другие вторичные имена, начинающиеся с S в каталоге rc2.d.

Это символизирует необходимость запускать (start) службы postfix и postgresql при переключении системы на уровень исполнения № 2 и уничтожать (kill) процессы этих служб при переключении системы на уровень исполнения № 6.

Типы файлов. Ссылки. Файловые системы.

Лекция 3

Типы файлов

Файлы как единицы обеспечения доступа к данным различаются операционной системой по типам, указывающим источник информации. Обычные (*regular*) файлы и каталоги (*directory*) обеспечивают сохранение информации на тех или иных носителях.

Специальные файлы устройств (*special device file*) позволяют обмениваться информацией с тем или иным аппаратным устройством ввода-вывода, а именованные каналы и файловые сокеты предназначены для обмена информацией между процессом одной программы и процессами других программ.

В примере из листинга 3.1 в полном (-l, *long*) выводе команды `ls` проиллюстрирован признак типа файла.

Символом - обозначается обычный файл, символом b или c — специальные файлы блочного (*block*) или символьного (*character*) устройства, символом p — именованный канал (*pipe*), символом s — сокет (*socket*), а символом l — символьическая ссылка (*link*).

Листинг 3.1. Признак типа файлов ls

```
finn@ubuntu:~$ ls -l /bin/ls /dev/sda /dev/tty /sbin/halt  
-rwxr-xr-x 1 root root 142144 сен 5 13:38 /bin/ls  
brw-rw--- 1 root disk 8, 0 ноя 17 03:31 /dev/sda  
crw-rw-rw- 1 root tty 5, 0 ноя 17 12:18 /dev/tty  
lrwxrwxrwx 1 root root 14 ноя 13 00:20 /sbin/halt -> /bin/systemctl  
finn@ubuntu:~$ ls -l /run/initctl /run/udev/control  
prw..... 1 root root 0 ноя 17 03:30 /run/initctl  
srw..... 1 root root 0 ноя 17 03:30 /run/udev/control
```

Обычные файлы

Обычные файлы содержат пользовательскую информацию: текст, изображения, звук, видео и прочие данные в виде набора байтов (см. на рис. 3.2, листинг 3.2).

За структуру содержания и имена обычных файлов ответственны прикладные программы, а операционная система не накладывает никаких ограничений

Листинг 3.2 Содержание обычных файлов

```
finn@ubuntu:~$ file /usr/share/man/man1/file.1.gz
```

```
/usr/share/man/man1/file.1.gz: gzip compressed data, max compression, from Unix,  
original size modulo 2A32 21484
```

```
finn@ubuntu:~$ file /etc/passwd
```

```
/etc/passwd: ASCII text
```

```
finn@ubuntu:~$ file /bin/ls
```

```
/bin/ls: ELF 64-bit LSB pie executable, X86-64, version 1 (SYSV), dynamically linked,  
interpreter /lib64/ld-linux-x86-64.so.2,
```

```
BuildID[sha1]=2f15ad836be3339dec0e2e6a3c637e08e48aacbd, for GNU/Linux 3.2.0,  
stripped
```

```
finn@ubuntu:~$ file /usr/share/sounds/alsa/Noise.wav
```

```
/usr/share/sounds/alsa/Noise.wav: RIFF (little-endian) data, WAVE audio, Microsoft PCM,  
16 bit, mono 48000 Hz
```

```
finn@ubuntu:~$ file /usr/share/backgrounds/Sky_Sparkles_by_Joe_Thompson.jpg
```

```
/usr/share/backgrounds/Sky_Sparkles_by_Joe_Thompson.jpg: JPEG image data, JFIF  
standard 1.01, aspect ratio, density 1x1, segment length 16, progressive, precision 8,  
3840x2160, components 3
```

Создать обычный файл можно при помощи любой программы, сохраняющей информацию в файл, например посредством текстовых редакторов `vi`, `nano` или `mcedit`.

Для создания пустого обычного файла можно воспользоваться командой `touch` — см. листинг 3.5.

Для удаления обычного файла предназначается команда `rm` — См. листинг 3.6.

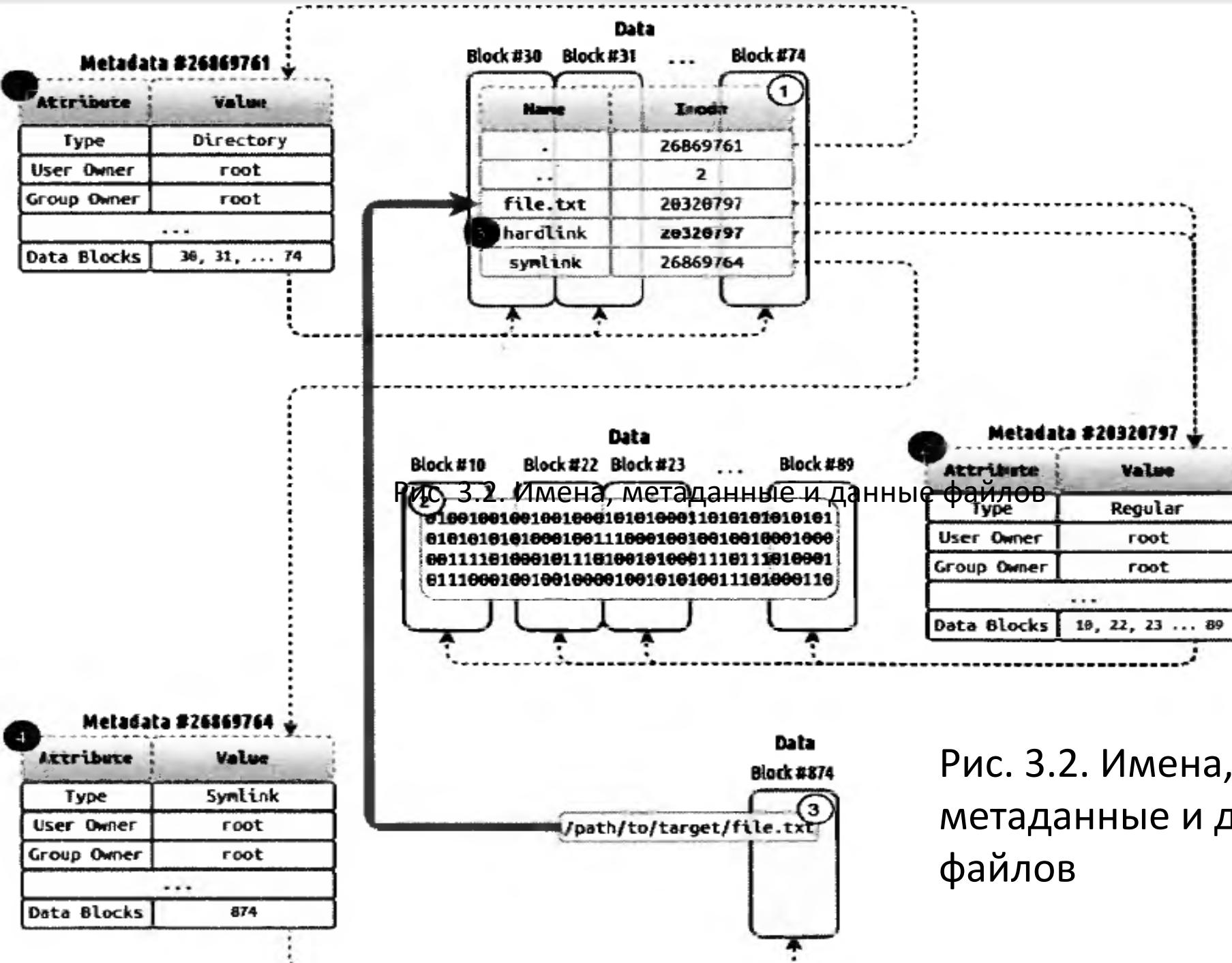


Рис. 3.2. Имена,
метаданные и данные
файлов

Каталоги

Файлы-каталоги, в отличие от обычных файлов, имеют служебное для операционной системы содержимое — таблицу имен (см. на рис. 3.2) файлов и соответствующих им номеров индексных дескрипторов (inode, index node), проиллюстрированных в листинге 3.3.

Листинг 3.3. Имена и номера индексных дескрипторов файлов finn@ubuntu:~\$ ls -at

```
20332580 . 20318930 .bash_logout 20320866 examples.desktop
```

```
20316161 .. 20320868 .bashrc 20320867 .profile
```

```
20320797 .bash_history 20332712 .cache
```

Каждый индексный дескриптор содержит метаданные (см. на рис. 3.2) — список стандартных свойств файла, в том числе указывающих местоположение данных файла (набора блоков) на файловой системе.

Полный набор метаданных (листинг 3.4) позволяет получить команда stat, включая размер файла , количество занимаемых блоков на диске , тип файла , номер индексного дескриптора , права доступа , владельцев и пр.

Листинг 3.4. Метаданные файлов

```
finn@ubuntu:~$ stat .profile
```

Файл: .profile

Размер: 807 Блоков: 8 Блок В/В: 4096 обычный файл

Устройство: 802h/2050d Inode: 393433 Ссылки: 1

Доступ (0644/-rw-r--r--)Uid: (1001/ fin.y Gid: (1001/ firm)

Доступ: 2019-11-17 15:14:31.673047212 +0300

Модифицирован: 2019-11-13 00:25:26.723714502 +0300

Изменён: 2019-11-13 00:25:26.723714502 +0300

Создан: -

Для создания каталогов предназначена команда `mkdir`, а для удаления — `rmdir`, при этом удалению подлежат только пустые каталоги (см. листинг 3.8).

Имена, данные, метаданные и индексные дескрипторы

Каждый раз, когда используется путевое (абсолютное или относительное) имя файла, производится итеративный поиск файла в дереве путем последовательного разбора на имя (по которому можно найти метаданные и данные) первого каталога пути, содержащего, в свою очередь, имя второго каталога пути, который содержит имя третьего и т. д., пока в конце поиска не будут найдены имя, метаданные и данные указанного файла.

Такая ссылочность позволяет сформировать удобную древовидную структуру для каталогизации файлов (называемую деревом каталогов), однако сам поиск является относительно длительной операцией.

Ссылки

Листинг 3.5 Жесткая ссылка

```
finn@ubuntu:~$ touch readme
```

```
finn@ubuntu:~$ ls -li readme
```

```
20318653 -rw-r--r-- 1 finn finn 0 апр. 1 01:22 readme
```

```
finn@ubuntu:~$ ln readme readme.txt
```

```
finn@ubuntu:~$ touch README
```

```
flnn@ubuntu:~$ ls -li readme readme.txt README
```

```
20318653 -rw-r--r-- 2 finn finn 0 апр. 1 01:22 readne
```

```
20319121 -rw-r--r-- 1 finn finn 0 апр. 1 01:23 README
```

```
20318653 -rw-r--r-- 2 finn finn 0 апр. 1 01:22 readne.txt
```

Более того, оба имени являются равнозначными, и нет возможности узнать, какое из них создано первым, из чего нужно заключить, что первое и единственное имя файла уже является его жесткой ссылкой (на номер индексного дескриптора).

При добавлении файлу нового имени (жесткой ссылки) в его метаданных увеличивается счетчик количества имен (см. в листинге 3.6), а при удалении файла сначала удаляется имя и уменьшается счетчик количества имен , и только при удалении последнего имени высвобождаются метаданные и данные файла.

Листинг 3.6 Счетчик файлов

```
finn@ubuntu:~$ ln readme read.me
```

```
finn@ubuntu:~$ ls -li read*
```

```
20318653 -rw-r--r-- 0 3 finn finn 0 anp.1 01:22 readme
```

```
20318653 -rw-r--r-- 3 finn finn 0 anp.1 01:22 read.me
```

```
20318653: -rw-r--r-3 finn finn 0 anp.1 01:22 readme.txt
```

```
finn@ubuntu:~$ m readme
```

```
finn@ubuntu:~$ ls -li read*
```

```
20318653 -rw-r--r-- ② 2 finn finn 0 anp.1 01:22 read.me
```

```
20318653 -rw-r-r-2 finn finn 0 anp.1 01:22 readme.txt
```

```
finn@ubuntu:~$ rn readne.txt finn@ubuntu:~$ ls -li read*
```

```
20318653 -rw-r--r-- ① 1 finn finn 0 anp.1 01:22 read.me
```

Нужно заметить, что удаление файла — двухшаговая операция, состоящая из удаления имени файла, а затем — удаления метаданных (и высвобождения блоков, занимавшихся этим файлом).

Удаление метаданных файла не выполняется вообще, если у файла еще остались имена (жесткие ссылки), и не происходит сразу, если файл открыт каким-либо процессом.

Метаданные и блоки, занимаемые файлом, высвобождаются только при закрытии этого файла всеми открывшими его процессами, что проиллюстрировано в примере из листинга 3.7.

Команда `df`(`D` измеряет доступное (свободное, `disk free`) место на файловой системе указанного файла, тогда как команда `du`, наоборот, измеряет занимаемое (`disk usage`) указанным файлом место на его файловой системе.

Листинг 3.7. Удаление открытого файла

```
finn@ubuntu:~$ df -h .
```

Файл.система Размер Использовано Дост Использовано% Смонтировано в
/dev/mapper/ubuntu - root 455G (400G 32G** 93% /

```
finn@ubuntu:~$ du -sh astra-linux-l.3-special-edition-smolensk-disk3-devel.iso  
2,8G astra-linux-l.3-special-edition-snolensk-disk3-devel.iso
```

```
finn@ubuntu:~$ m astra-linux-l.3-special-edition-snolensk-disk3-devel.iso
```

```
finn@ubuntu:~$ df -h .
```

Файл.система Размер Использовано Дост Использовано% Смонтировано в
? /dev/mapper/ubuntu-root 455G 400G 32G*» 93% /

```
finn@ubuntu:~$ lsof astra-linux-l.3-special-edition-snolensk-disk3-devel.iso COMMAND PID USER FD TYPE  
DEVICE SIZE/OFF NODE NAME
```

```
fuseiso 16925 finn 3r REG 252,0 2947385344 20316584 astra-linux-1.3-special-edition-  
Smolensk-disk3-devel.iso
```

```
finn@ubuntu:~$ kill 16925
```

```
finn@ubuntu:~$ df -h .
```

Файл.система Размер Использовано Дост Использовано% Смонтировано в
! /dev/mapper/ubuntu-root 455G 397G 35G*» 92% /

Специальные имена текущего и родительского каталогов на поверху тоже оказываются жесткими ссылками, поэтому у любого каталога по крайней мере два имени — свое «собственное» в родительском и специальное . в самом себе, а у каталогов с подкаталогами еще и имена в каждом из дочерних (листинг 3.8).

Листинг 3.8. Имена каталогов

```
finn@ubuntu:~$ mkdir folder
```

```
finn@ubuntu:~$ ls -ldt folder
```

```
20357139 drwxr-xr-x. 2 finn finn 4096 апр.
```

```
finn@ubuntu:~$ cd folder
```

```
finn@ubuntu:~/folder$ ls -lai итого 8
```

```
20357139 drwxr-xr-x 2 finn finn 4096 апр.
```

```
20332580 drwxr-xr-x 4 finn finn 4096 апр.
```

```
inn@ubuntu:~/folder$ mkdir child
```

```
finn@ubuntu:~/folder$ cd child
```

```
finn@ubuntu:~/folder/child$ ls -lai
```

```
итого 8
```

Существенным ограничением жесткой ссылки в дереве каталогов, куда смонтирована более чем одна файловая система, является локальность жесткой ссылки в пределах своей файловой системы в силу локальной значимости номеров индексных дескрипторов.

Так как на каждой новой файловой системе номера индексных дескрипторов начинают нумероваться с нуля, то жесткая ссылка всегда указывает на метаданные файла в «своей» файловой системе и не может указывать на метаданные файла в «чужой» файловой системе общего дерева каталогов.

Для преодоления этого ограничения служит символьическая ссылка `symlink`, являющаяся самостоятельным служебным типом (см. на рис. 3.2) и содержащая путевое имя (см. на рис. 3.2 и в листинге 3.9) к целевому файлу.

Листинг 3.9. Символическая ссылка

```
finn@ubuntu:~$ ln -s read.me readme.lst
```

```
finn@ubuntu:~$ ls -li read*
```

```
20318653 -rw-r--r-- 1 finn finn 0 апр. 1 01:22 read.me
```

```
20319944 lrwxrwxrwx 1 finn finn 7 апр. 2 00:03 readme.lst -> read.me
```

В случае с символической ссылкой при удалении целевого файла сама ссылка будет указывать в никуда и называться «сиротой» (orphan). Попытка прочитать (листинг 3.10) такую ссылку приводит к странным на первый взгляд результатам: файл «существует» для команды ls, но команда просмотра содержимого cat говорит об обратном.

Ничего удивительного, если помнить, что ls работает с именами файлов, а cat — с их данными (которые действительно не существуют).

Листинг 3.10 Сиротская ссылка

```
finn@ubuntu:~$ rm read.me
```

```
finn@ubuntu:~$ ls read*
```

```
readme.1st
```

```
ftnn@ubuntu:~$ cat readme.1st
```

```
cat: readme.1st: Нет такого файла или каталога
```

```
finn@ubuntu:~$ ls -l read*
```

```
lrwxrwxrwx 1 finn finn 7 апр. 2 00:03 readme.1st -> read.me
```

```
finn@ubuntu:~$ cat read.me
```

```
cat: read.me: Нет такого файла или каталога
```

Символические ссылки могут ссылаться на имена друг друга неограниченное количество раз, а при попытке использовать одну из таких ссылок будут прочитаны данные файла, последнего в цепочке ссылок.

По ошибке можно даже закольцевать (листинг 3.11) две или более ссылок, с чем разберется операционная система при чтении одной из ссылок.

Листинг 3.11 Кольцевые ссылки

```
finn@ubuntu:~$ ln -s readme.lst read.me
```

```
finn@ubuntu:~$ ls -l read*
```

```
lrwxrwxrwx 1 finn finn 10 апр. 2 00:41 read.me -> readme.lst
```

```
lrwxrwxrwx 1 finn finn 7 апр. 2 00:03 readme.lst -> read.me
```

```
finn@ubuntu:~$ cat read.me
```

```
cat: read.me: Слишком много уровней символьных ссылок
```

Основным назначением символьических (и изначально, жестких) ссылок является «множественная каталогизация» файлов, т. е. различные наборы разных имен одних и тех же данных.

Типичным примером использования ссылок является организация boot сценариев запуска системных служб при старте операционной системы.

Сами сценарии располагаются в каталоге `/etc/inlt.d`, а в каталогах `/etc/rcS.d`, `/etc/rc0.d`, ..., `/etc/rc6.d` размещены символические ссылки на эти сценарии, которые должны быть запущены с определенными параметрами при переключении состояния системы между так называемыми «уровнями исполнения» (листинг 3.12).

Листинг 3.12 Сценарий запуска служб и их каталогизация по уровням исполнения

```
flnn@ubuntu:~$ ls -l /etc/rc?.d
```

```
/etc/rc2.d:
```

```
lrwxrwxrwx 1 root root 20 ноя 13 00:16 /etc/rc2.d/S19postgresql -> ../init.d/postgresql
```

```
lrwxrwxrwx 1 root root 17 ноя 13 00:16 /etc/rc2.d/S20postfix -> ../init.d/postfix
```

```
/etc/rc6.d:
```

```
lrwxrwxrwx 1 root root 17 ноя 13 00:16 /etc/rc6.d/K20postfix -> ../init.d/postfix
```

```
lrwxrwxrwx 1 root root 17 ноя 13 00:16 /etc/rc6.d/K21postgresql -> ../init.d/postgresql
```

В этом примере сценарии postfix и postgresql имеют вторичные имена, начинающиеся с K в каталоге rc6.d, и другие вторичные имена, начинающиеся с S в каталоге rc2.d.

Это символизирует необходимость запускать (start) службы postfix и postgresql при переключении системы на уровень исполнения № 2 и уничтожать (kill) процессы этих служб при переключении системы на уровень исполнения № 6.

Специальные файлы устройств

Специальные файлы устройств предназначены для ввода данных с аппаратных устройств и вывода данных на них.

Настоящую работу по вводу и выводу данных проделывает драйвер устройства, а специальные файлы (листинг 3.13) играют роль своеобразных «порталов» связи с драйверами.

Различают символьные и блочные специальные файлы устройств, у которых минимальной единицей обмена информацией с драйверами является блок (обычно размером 512 байт) или символ (1 байт), соответственно.

Листинг 3.13 Специальные файлы устройств

```
finn@ubuntu:~$ ls -l /dev/sd* /dev/input/mouse* /dev/video* /dev/snd/pcm*
```

сущ-пн----	1	root	input	13 ①, 32 ②	ноя 17 03:31	/dev/input/mouse0
сущ-пн----	1	root	input	13, 33	ноя 17 03:31	/dev/input/mouse1
бущ-пн----	1	root	disk	8, 0	ноя 17 03:31	/dev/sda
бущ-пн----	1	root	disk	8, 1	ноя 17 03:31	/dev/sda1
бущ-пн----	1	root	disk	8, 5	ноя 17 03:31	/dev/sda5
сущ-пн----+	1	root	audio	116, 3	ноя 17 15:03	/dev/snd/pcmC0D0c
сущ-пн----+	1	root	audio	116, 2	ноя 17 16:01	/dev/snd/pcmC0D0p
сущ-пн----+	1	root	audio	116, 4	ноя 17 03:31	/dev/snd/pcmC0D1c
сущ-пн----+	1	root	video	81, 0	ноя 17 03:31	/dev/video0

Все драйверы ядра пронумерованы главными (мажорными, major) числами, а аппаратные устройства, находящиеся под их управлением, — дополнительными (минорными, minor) числами.

Например, все IDE-диски работают под управлением драйвера `hd`, имеющего 3 major (первичный контроллер) и 22 major (вторичный контроллер), и нумеруются как 0 minor (мастер-диск) и 64 minor (слэйв-диск).

Аналогично, SCSI-диски работают под управлением драйвера `sd`, имеющего 8 major, и нумеруются 0 minor (первый диск), 16 minor (второй диск) и т. д.

Основной характеристикой специальных файлов устройств является пара чисел `major`, `minor` (иногда называемых характеристическими числами), привязывающая их к конкретному драйверу и управляемому им устройству.

Имена специальных файлов и их местоположение в дереве каталогов не имеют никакого значения, но по соглашению `MAKEDEV` их принято располагать в каталоге `/dev` и именоватьозвучно именам драйверов.

Специальными файлами дисковых устройств пользуются программы, управляющие структурами самого носителя, например таблицами разделов `fdisk` и `parted` (листинг 3.14), или механикой накопителя `eject` (листинг 3.15), или файловыми системами разделов носителя `mount`, `fsck`, `mkfs` и пр.

Листинг 3.14. Чтение таблицы разделов диска

```
finn@ubuntu:~$ sudo fdisk -l /dev/sda
```

Диск /dev/sda: 500.1 Гб, 500107862016 байт

255 головок, 63 секторов/треков, 60801 цилиндров, всего 976773168 секторов

Units = секторы of 1 * 512 = 512 bytes

Размер сектора (логического/физического): 512 байт / 4096 байт

I/O size (minimum/optimal): 4096 bytes / 4096 bytes

Идентификатор диска: 0x000c8d62

Устройство	Загр	Начало	Конец	Блоки	Id	Система
/dev/sda1	*	2048	499711	248832	83	Linux
/dev/sda2		501758	976771071	488134657	5	Расширенный
Partition 2 does not start on physical sector boundary.						
/dev/sda5		501760	976771071	488134656	8e	Linux LVM

Листинг 3.15. Открытие лотка DVD

```
finn@ubuntu:~$ ls -l /dev/dvd  
lrwxrwxrwx 1 root root 3 марта 27 14:54 /dev/dvd -> sr0  
finn@ubuntu:~$ eject /dev/dvd
```

Особенное место занимают драйверы терминалов — окончных устройств для взаимодействия с пользователями.

Аппаратные терминалы (например, VT100), подключающиеся посредством последовательного интерфейса RS232, доступны при помощи специальных файлов /dev/ttySN драйвера приемопередатчика последовательного порта ttyS.

Виртуальные терминалы, реализуемые консолью (стандартной клавиатурой и дисплеем в алфавитно-цифровом режиме — см. рис. 2.3), доступны при помощи специальных файлов /dev/ttyN (листинг 3.16) драйверов консоли console_ioctl и tty_ioctl.

Листинг 3.16 Специальные файлы терминалов

```
finn@ubuntu:~$ ls -la /dev/tty?  
crw--w--- 1 root tty 4, 0 ноя 17 03:31 /dev/tty0  
crw.....1 finn tty 4, 1 ноя 17 15:03 /dev/tty1  
crw.....1 jake tty 4, 2 ноя 17 13:27 /dev/tty2  
crw.....1 marceline tty 4, 3 ноя 17 15:14 /dev/tty3  
crw.....1 icekIng tty 4, 4 ноя 17 03:31 /dev/tty4  
crw..... 1 bubblegum tty 4, 5 ноя 17 03:31 /dev/tty5
```

Именно эти специальные файлы терминалов используют команды write и wall для отсылки сообщений пользователям, зарегистрировавшимся в системе, setFont — для смены шрифтов, chvt(1) — для переключения между терминалами, а setleds — для управления светодиодами клавиатуры (листинг 3.17).

Листинг 3.17. Включение светодиодов CAPS LOCK и SCROLL LOCK

```
finn@ubuntu:~$ tty  
/dev/tty1  
finn@uburttu:~$ setleds -L +num +scroll
```

Кроме специальных файлов настоящих аппаратных устройств, в арсенале Linux имеются особые файлы псевдоустройств, такие как `/dev/null`, `/dev/full` и `/dev/zero`, симулирующих всегда пустое, всегда полное и бесконечно нулевое устройство, см. `null`.

Всегда пустое устройство `/dev/null` широко применяется на практике в качестве «подавляющего» стока информации при перенаправлениях, а бесконечно нулевое устройство `/dev/zero` зачастую используют в качестве источника для получения нулевых файлов нужной величины.

Псевдоустройства `/dev/random` и `/dev/urandom` организуют доступ к ядерному генератору случайных и псевдослучайных чисел `random`, основанных на внешних событиях периферийных устройств.

Потоки случайных чисел используются в качестве источников энтропии для криptoалгоритмов, в частности библиотекой W:[OpenSSL], или служат команде `shred` источником случайных байтов для надежного стирания данных.

Именованные каналы и файловые сокеты

Именованные каналы и файловые сокеты являются простейшими средствами межпроцессного взаимодействия (IPC, InterProcess Communication) и служат программам для обмена информацией между собой.

Разные программы выполняются в рамках различных процессов (изолированных друг от друга), поэтому для общения нуждаются в специальных средствах взаимодействия.

Таким средством могли бы стать обычные файлы, но их основное назначение состоит в сохранении информации на каком-либо накопителе, что будет при обмене информацией сопряжено с накладными расходами, например задержками записи/чтения дискового (т. е. механического) носителя.

Предоставить процессам возможность использовать файловые операции для эффективного взаимодействия между собой призваны именованные каналы (named pipe) pipe, они же FIFO-файлы (first in first out) fifo и файловые сокеты (socket) unix.

Каналы и сокеты используют для передачи данных от процесса к процессу оперативную память ядра операционной системы, а не память накопителя, как обычные файлы.

Основное отличие именованного канала от сокета состоит в способе передачи данных.

Через именованный канал организуется односторонняя (симплексная) передача без мультиплексирования, а через сокет — двунаправленная (дуплексная) мультиплексированная передача.

Именованный канал обычно используют при взаимодействии процессов по схеме «поставщик — потребитель» (producer-consumer), когда один потребитель принимает информацию от одного поставщика (на самом деле от разных, но в различные моменты времени).

Например, программы `halt`, `shutdown`, `reboot`, `poweroff` и `telinit` передавали ранее посредством именованного канала `/dev/lntctl` команды перезагрузки, выключения питания и другие диспетчеру `init`, который и выполнял соответствующие действия.

Сокет используют при взаимодействии по схеме «клиент — сервер» (client-server), т. е. один сервер принимает и отправляет информацию от многих и ко многим (одновременно) клиентам.

Например, в целях централизованного сбора событийной информации разнообразные службы операционной системы (в частности, служба периодического выполнения заданий cron, служба печати cupsd и даже команда logger) передают посредством файлового сокета /dev/log (в современный момент является символической ссылкой на актуальный /run/systemd/journal/dev-log) сообщения о происходящих событиях службе журнализации, представленной в современных системах systemd-journald.

Файловые дескрипторы

Основными операциями, предоставляемыми ядром операционной системы программам (а точнее — процессам) для работы с файлами, являются системные вызовы `open`, `read`, `write` и `close`.

В соответствии со своими именами эти системные вызовы предназначены для открытия и закрытия файла, для чтения из файла и записи в файл.

Дополнительный системный вызов `ioctl` (`i`nput `o`utput `c`ontrol) используется для управления драйверами устройств и, как следствие, применяется в основном для специальных файлов устройств.

При запросе процесса на открытие файла системным вызовом `open` производится его однократный (относительно медленный) поиск имени файла в дереве каталогов и для запросившего процесса создается так называемый файловый дескриптор (описатель, от англ. `descriptor`).

Файловый дескриптор «содержит» информацию, описывающую файл, например индексный дескриптор `inode` файла на файловой системе, номера `major` и `minor` устройства, на котором располагается файловая система файла, режим открытия файла и прочую служебную информацию.

При последующих операциях `read` и `write` доступ к самим данным файла происходит с использованием файлового дескриптора (что исключает медленный поиск файла в дереве каталогов).

Файловые дескрипторы пронумерованы и содержатся в таблице открытых процессом файлов, которую можно получить (листинг 3.18) при помощи диагностической программы IsoF).

Наоборот, получить список процессов, открывших тот или иной файл, можно при помощи программ Isof и fuser, что бывает полезно для идентификации программ, «занявших» файловую систему, подлежащую отмонтированию (см. ★ в листинге 3.21).

Листинг 3.18. Таблица файловых дескрипторов

В первом примере из листинга 3.18 показано получение списка файловых дескрипторов (столбец FD) процесса, командного интерпретатора bash пользователя finn, на котором файловый дескриптор номер **1** описывает открытый, на чтение и запись и специальный символьный CHR файл устройства /dev/pts/2.

① finn@ubuntu:~\$ lsolf -p \$\$¹

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE/OFF	NODE	NAME

bash	17975	finn	1u	①	CHR	136,2	0t0	5 /dev/pts/2

Листинг 3.18 Продолжение

Во втором примере показано получение информации о процессах, открывших файловый сокет unix с именем /run/systemd/journal/dev-log (файловый дескриптор номер 6 на чтение и запись u) и обычный файл REG с именем /var/log/syslog (файловый дескриптор номер 8 на запись w).

```
② root@ubuntu:~# ls -la /dev/log
lrwxrwxrwx 1 root root 28 ноя 17 03:30 /dev/log -> /run/systemd/journal/dev-log
root@ubuntu:~# lsof /run/systemd/journal/dev-log
COMMAND   PID USER   FD   TYPE      DEVICE SIZE/OFF NODE NAME
systemd     1 root    35u  unix 0xfffff964892232400      0t0 13321 /run/.../dev-log ...
systemd-j 285 root    6u  ● unix 0xfffff964892232400      0t0 13321 /run/.../dev-log ...

root@ubuntu:~# fuser /run/systemd/journal/dev-log
/run/systemd/journal/dev-log:    1  285 •

root@ubuntu:~# ps p 285
PID TTY      STAT   TIME COMMAND
285 ?        S<s   0:04 /lib/systemd/systemd-journald

root@ubuntu:~# lsof /var/log/syslog
COMMAND   PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
rsyslogd 606 syslog    8w  ●  REG      8,2 1920291 131082 /var/log/syslog

root@ubuntu:~# ps up 606
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
syslog     606  0.0  0.1 224360  4584 ?      Ssl  ноя17  0:00 /usr/sbin/rsyslogd -n ...
```

Пронаблюдать за использованием системных вызовов файлового программного интерфейса в момент выполнения программам позволяет системный трассировщик strace (листинг 3.19).

Листинг 3.19. Трассировка файлового программного интерфейса

```
finn@ubuntu:~$ date
Пн ноя 18 00:13:53 MSK -> 2019
finn@ubuntu:~$ strace -fe open,openat,close,read,write,ioctl date
...
...
...
-> openat(AT_FDCWD, "/etc/localtime", O_RDONLY|O_CLOEXEC) = 3
...
...
...
finn@ubuntu:~$ file /etc/localtime
/etc/localtime: symbolic link to /usr/share/zoneinfo/Europe/Moscow
finn@ubuntu:~$ file /usr/share/zoneinfo/Europe/Moscow
/usr/share/zoneinfo/Europe/Moscow: timezone data, version 2, 17 gmt time flags, 17 std time
flags, no leap seconds, 78 transition times, 17 abbreviation chars
```

Листинг 3.19. Продолжение

```
finn@ubuntu:~$ ls -la /dev/dvd
```

```
lrwxrwxrwx 1 root root 3 ноя 17 22:35 /dev/dvd -> sr0
```

```
finn@ubuntu:~$ strace -fe open,openat,close,read,write,ioctl eject
```

```
... ... ...
openat(AT_FDCWD, "/dev/sr0", O_RDWR|O_NONBLOCK) = 3 
```

```
→ ioctl(3, CDROMEJECT, 0x01) = 0
```

```
close(3) = 0
```

```
finn@ubuntu:~$ strace -fe open,openat,read,write,close,ioctl setleds -L +num +scroll
```

```
... ... ...
ioctl(0, KDGKBLED, 0x7ffc1ced3d27) = 0
```

```
ioctl(0, KDGTLLED, 0x7ffc1ced3d26) = 0
```

```
ioctl(0, KDSSETLED, 0x3) = 0
```

Предположив, что программа date показывает правильное московское время, потому что узнаёт заданную временную зону MSK из некоего конфигурационного файла операционной системы, при трассировке ее работы можно установить его точное имя — /etc/localtime (оказавшимся символьической ссылкой на /usr/share/zoneinfo/Europe/Moscow).

Аналогично предположив, что программа eject открывает лоток привода CD/DVD при помощи специального файла устройства, при трассировке можно узнать имя файла — /dev/sr0, номер файлового дескриптора при работе с файлом — 3 и команду CDROMEJECT соответствующего устройству драйвера ioctl_list. Трассировка команды setleds показывает, что она вообще не открывает никаких файлов, но пользуется файловым дескриптором 6 так называемого стандартного потока ввода (прикрепленного к текущему терминалу) и командами KDGTLLED и KDSETLED драйвера консоли console_ioctl.

Файловые системы

Файловые системы и процедура монтирования

Доступ к информации организуется при помощи файлов, упорядоченных в единое «воображаемое» дерево каталогов, тогда как «настоящими» источниками данных являются файловые системы — структуры, решающие задачи хранения информации.

Отображение множества файловых систем в единое дерево каталогов реализуется посредством процедуры монтирования.

Таким образом, все, что наблюдается в дереве каталогов, в реальности размещается на файловых системах.

Состав дерева каталогов показывает команда `mount`, равно как и присоединяет к нему — монтирует очередную файловую систему.

Отсоединяет (отмонтирует) файловую систему от дерева каталогов команда `umount`, но только при условии, что ни один файл на этой файловой системе не используется никакой программой (а правильнее — никаким процессом) операционной системы.

В примере на рис. 3.3 и в листинге 3.20 иллюстрируются: файловая система W:[ext4], располагающаяся на дисковом накопителе, идентифицируемая файлом устройства /dev/sda2, и смонтированная непосредственно в корень дерева каталогов ; файловая система vfat flash-накопителя в на устройстве /dev/sdb1, смонтированная в /media/flash; файловая система W: [ISO 9660] CD-диска на устройстве /dev/sr6, смонтированная в /media/cdron.

Кроме этого, в дерево каталогов смонтированы две псевдофайловые системы `/proc` и `/sysfs`, считающие из оперативной памяти ядра операционной системы информацию о процессах, обнаруженных устройствах, загруженных драйверах и предоставляющих «файловый» доступ к ней.

Листинг 3.20. Состав дерева каталогов

```
finn@ubuntu:~$ mount
/dev/sda2 on / type ext4 (rw,relatime,errors=remount-ro)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
...
...
...
/dev/sdb1 on /media/flash type vfat (rw,...)
/dev/sr0 on /media/cdrom type iso9660 (ro,...)
```

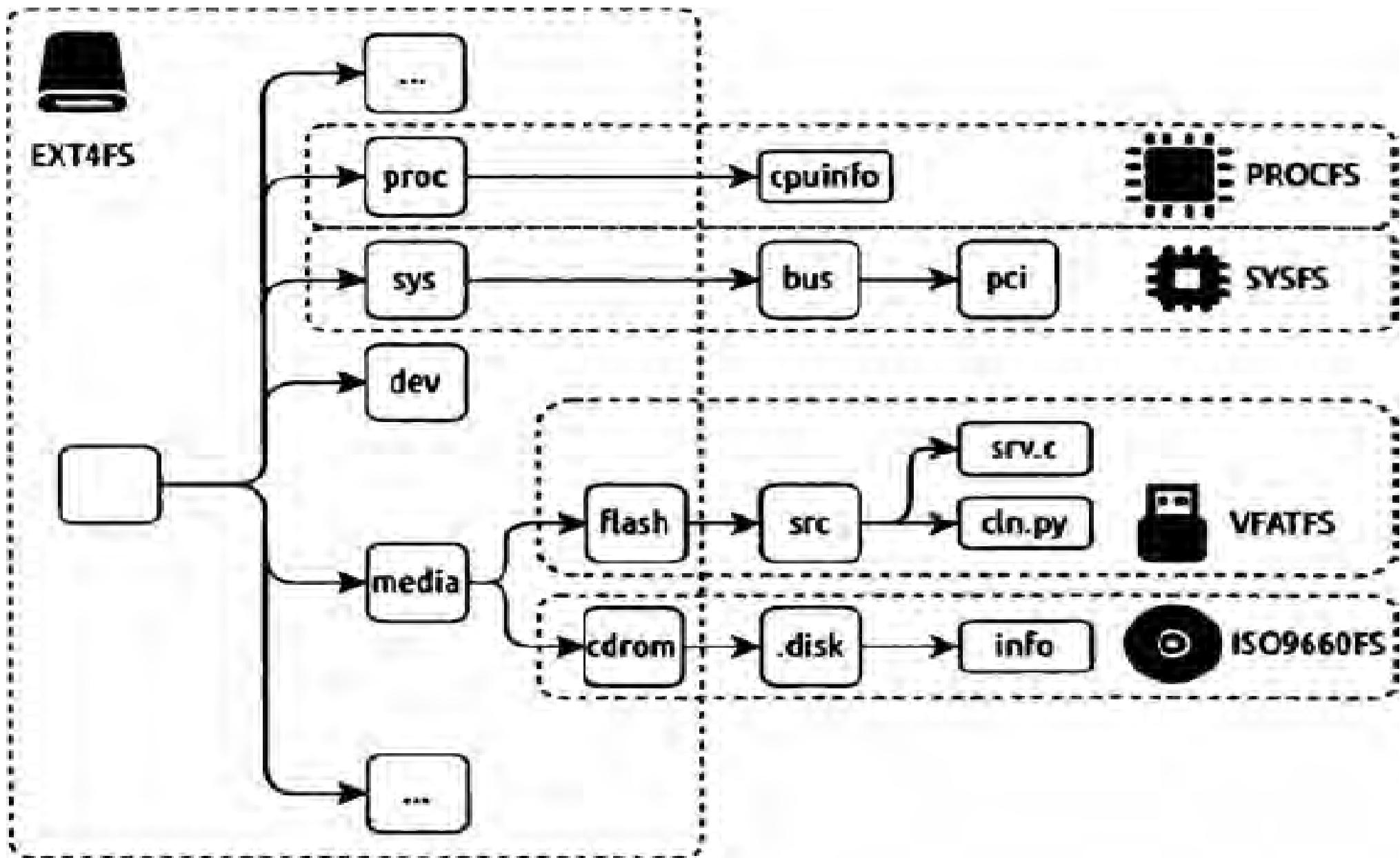


Рис. 3.3. Монтирование файловых систем

Несмотря на то, что в современных дистрибутивах Linux обнаружение и процедуры монтирования файловых систем автоматизированы, операции монтирования/размонтирования могут быть произведены вручную (листинг 3.21).

Листинг 3.21 Процедуры монтирования и размонтирования

```
finn@ubuntu:~$ mount /dev/dvd /media/cdrom
```

```
mount: только root может сделать это
```

```
finn@ubuntu:~$ sudo mount /dev/dvd /media/cdrom
```

```
mount: /media/cdrom: ВНИМАНИЕ: устройство защищено от записи, смонтировано только для чтения.
```

```
finn@ubuntu:~$ mount
```

```
... ... ... ...  
/dev/dvd on /media/cdrom type iso9660 (ro,...)
```

```
... ... ... ...
```

```
finn@ubuntu:~$ cat /media/cdrom/.disk/info
```

```
Ubuntu 19.10 "Eoan Ermine" - Release amd64 (20191017)
```

```
finn@ubuntu:~$ umount /media/cdrom
```

```
umount: /media/cdrom: umount failed: Операция не позволена.
```

Листинг 3.21 Продолжение

```
finn@ubuntu:~$ sudo umount /media/cdrom
```

```
finn@ubuntu:~$ cat /media/cdrom/.disk/info
```

```
cat: /media/cdrom/.disk/info: Нет такого файла или каталога
```

```
finn@ubuntu:~$ sudo umount /proc
```

```
umount: /proc: target is busy.
```

```
finn@ubuntu:~$ mount /dev/sdc1 /media/flash
```

```
finn@ubuntu:~$ mount
```

```
***
```

```
***
```

```
***
```

```
***
```

```
/dev/sdc1 on /media/flash type vfat (rw,...)
```

```
***
```

```
***
```

```
***
```

```
***
```

Дисковые файловые системы

Разные файловые системы fs, как упоминалось ранее, предназначены для хранения информации на внешних носителях и преследуют различные цели, например обеспечивают надежное хранение при помощи журнала транзакций или быстрый поиск метаданных файла (среди множества каталогов, подкаталогов и других файлов) по его имени либо учитывают специфику свойств самого носителя и т. д.

В большинстве случаев до сих пор носителями информации являются магнитные или оптические диски, - благодаря чему файловые системы, размещаемые на них, зачастую называются «дисковыми» файловыми системами, даже если используются на твердотельных (flash) носителях.

Для магнитных дисков, характеризуемых возможностью чтения и записи блоков информации в произвольное место носителя (random access), в Linux на текущий момент времени используются «родные» файловые системы W:[Ext2], W:[Ext3] и W:[Ext4], специально разработанные W:[ReiserFS] и W:[Reiser4], а также заимствованные W: [XFS] и W:[JFS].

Для оптических CD/DVD-дисков, имеющих специфику записи в виде спиральной дорожки, применяются файловые системы W:[ISO 9660] и W:[udf]. Для USB-flash-накопителей в большинстве случаев используются заимствованные файловые системы W: [FAT] и W: [NTFS] в силу использования этих накопителей как мобильных средств переноса данных между разными компьютерами с различными операционными системами.

Сетевые файловые системы

Сетевые файловые системы, равно как и дисковые, обеспечивают хранение информации на внешнем носителе, которым в этом случае выступает файловый сервер (например, домашний NAS, Network Attached Storage), доступный по протоколу NFS (Network File System, W: [Network File System]), CIFS/SMB (Common Internet FileSystem или Server Message Block, W:[Server_Message_Block]) или им подобным. Одноименные файловые системы nfs и cifs/smb используются для монтирования файлов сервера в дерево каталогов клиента.

Таким образом, обычные (ничего не знающие ни про какие сетевые протоколы) программы, запускаемые в операционной системе клиента, используют файлы сетевого сервера точно так, как если бы они были размещены на локальных дисках, под управлением дисковых файловых систем.

В примере из листинга 3.22 программы avconv и avprobe, предназначенные для работы с «обычными» видеофайлами, используются для обработки записей сетевого видеорегистратора, видеофайлы которого доступны по протоколу NFS. Смонтированные при помощи сетевой файловой системы nfs в дерево каталогов файлы сетевого регистратора становятся никак неотличимы от файлов локальных дисковых файловых систем.

Листинг 3.22. Сетевая файловая система NFS

```
finn@ubuntu:~$ mount -t nfs 182.168.1.10:/share/video /mnt/nas/video
finn@ubuntu:~$ mount
...
182.168.1.10:/share/video on /mnt/nas/video type nfs (rw,...)
...
finn@ubuntu:~$ cd /mnt/nas/video/screenshots
finn@ubuntu:~$ ls
...
20140523142626.mp4
...
finn@ubuntu:~$ file 20140523142626.mp4
20140523142626.mp4: ISO Media, MPEG v4 system, version 2

finn@ubuntu:~$ avprobe 20140523142626.mp4
...
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from '20140523142626.mp4':
...
Duration: 00:00:07.94, start: 0.000000, bitrate: 11020 kb/s
Stream #0.0(eng): Video: h264 (High), yuv420p, 1920x1080 [PAR 1:1 DAR 16:9], 10843 kb/s,
50 fps, 50 tbr, 50k tbn, 100 tbc
...
```

Листинг 3.22 Продолжение

```
finn@ubuntu:~$ avconv -i 20140523142626.mp4 20140523142626.mkv
...
Output #0, matroska, to '20140523142626.mkv':
...
Stream #0.0(eng): Video: mpeg4, yuv420p, 1920x1080 [PAR 1:1 DAR 16:9], q=2-31, 200 kb/s,
1k tbn, 50 tbc
...
Stream mapping:
Stream #0:0 -> #0:0 (h264 -> mpeg4)
Stream #0:1 -> #0:1 (aac -> libvorbis)

Press ctrl-c to stop encoding

frame= 395 fps= 72 q=31.0 lsize= 2481kB time=7.94 bitrate=2561.5kbits/s dup=0 drop=1
video:2339kB audio:128kB global headers:4kB muxing overhead 0.457686%
```

Аналогично, в примере из листинга 3.23 геотеги файлов изображений сетевого видеорегистратора анализируются утилитой exiv2, предназначеннной для работы с «обычными» изображениями.

За счет файловой системы cifs и доступности видеорегистратора по протоколу CIFS его содержимое смонтировано в дерево каталогов так, словно сетевой регистратор является локальным дисковым накопителем

Листинг 3.21 Сетевая файловая система CIFS/SMB

```
finn@ubuntu:~$ mount -t cifs -o username=guest //182.168.1.10/share/photos /mnt/nas/photos
Password:
finn@ubuntu:~$ mount
//182.168.1.10/share/photos on /mnt/nas/video type cifs (rw,...)
...
finn@ubuntu:~$ cd /mnt/nas/photos
finn@ubuntu:~$ ls
DSC_0034.JPG  DSC_0043.JPG  DSC_0062.JPG  DSC_0074.JPG  DSC_0100.JPG  DSC_0189.JPG
...
finn@ubuntu:~$ exiv2 -p a DSC_0043.JPG
...
Exif.GPSInfo.GPSLatitudeRef          Ascii    2  North
Exif.GPSInfo.GPSLatitude            Rational   3  60deg 10' 3.479"
Exif.GPSInfo.GPSLongitudeRef        Ascii    2  East
Exif.GPSInfo.GPSLongitude          Rational   3  24deg 57' 23.294"
...
```

Специальные файловые системы

Развитие идеи файла как единицы обеспечения доступа к информации привело к тому, что абстракцию файловой системы перенесли и на другие сущности, доступ к которым стал организовываться в виде иерархии файлов.

Например, информацию о процессах, нитях и прочих сущностях ядра операционной системы и используемых ими ресурсах предоставляет программам виде файлов (!) псевдофайловая система `/proc`.

Таким же образом, информацию об аппаратных устройствах, обнаруженных ядром операционной системы на шинах PCI, USB, SCSI и пр., предоставляет псевдофайловая система `/sysfs`.

Различные утилиты, пользующиеся ядерной информацией, например показывающие нагрузку на операционную систему `uptime` или списки процессов и загруженных модулей (драйверов) ядра операционной системы — `ps` и `lsmod`, пользуются псевдофайловой системой `/proc`, в чем позволяет убедиться трассировка системных вызовов `open` (листинг 3.24).

Листинг 3.24. Псевдофайловая система

```
finn@ubuntu:~$ strace -fe open,openat uptime
...
...
...
openat(AT_FDCWD, "/proc/sys/kernel/osrelease", O_RDONLY) = 3
...
...
openat(AT_FDCWD, "/proc/uptime", O_RDONLY) = 3
...
...
openat(AT_FDCWD, "/proc/loadavg", O_RDONLY) = 4
...
...
15:42:32 up 12 days, 5:27, 7 users, load average: 0.48 ▶, 0.33, 0.32
finn@ubuntu:~$ cat /proc/uptime
1056774.23 1667210.55
finn@ubuntu:~$ cat /proc/loadavg
0.48 ▶ 0.33 0.32 1/623 14277
```

Аналогично, утилиты, показывающие список устройств на шинах PCI, USB и SCSI — `lspci`, `lsusb` и `lsscsi`, пользуются псевдофайловой системой `sysfs` (листинг 3.25).

Листинг 3.25 Псевдофайловая система `sysfs`

```
finn@ubuntu:~$ strace -fe open,openat lspci -nn
...
...
...
...
openat(AT_FDCWD, "/sys/bus/pci/devices/0000:00:02.0/resource", O_RDONLY) = 4
openat(AT_FDCWD, "/sys/bus/pci/devices/0000:00:02.0/irq", O_RDONLY) = 4
openat(AT_FDCWD, "/sys/bus/pci/devices/0000:00:02.0/vendor", O_RDONLY) = 4
openat(AT_FDCWD, "/sys/bus/pci/devices/0000:00:02.0/device", O_RDONLY) = 4
openat(AT_FDCWD, "/sys/bus/pci/devices/0000:00:02.0/class", O_RDONLY) = 4
openat(AT_FDCWD, "/sys/bus/pci/devices/0000:00:02.0/config", O_RDONLY) = 3
...
...
...
...
00:02.0 VGA compatible controller [0300 ↵]: Intel Corporation 2nd Generation Core Processor
Family Integrated Graphics Controller [8086 ↵:0116 ↵] (rev 09)
finn@ubuntu:~$ cat /sys/bus/pci/devices/0000:00:02.0/vendor
0x8086 ↵
finn@ubuntu:~$ cat /sys/bus/pci/devices/0000:00:02.0/device
0x0116 ↵
finn@ubuntu:~$ cat /sys/bus/pci/devices/0000:00:02.0/class
0x030000 ↵
```

Внеядерные файловые системы

Существование разных по своей сути файловых систем в едином дереве файлов и каталогов позволяет использовать для работы с его файлами «обычные» программы, ничего не знающие ни о местоположении, ни о свойствах самих источников информации этих файлов.

Таким же «файловым» образом можно организовать доступ к файлам внутри GZ/ZIP/...-архивов, файлам внутри ISO-образов CD/DVD-дисков, файлам зашифрованных каталогов, файлам других узлов сети, медиафайлам на медиаустройствах (плееры, смартфоны) и т. п.

Файловая абстракция в определенных случаях оказывается удобной даже для осуществления доступа к информации, организованной совершенно «нефайловым» образом — к данным таблиц и представлений внутри реляционных баз данных, к элементам и атрибутам внутри XML-файлов и пр.

Для реализации такого «файлового» доступа к произвольным источниками информации используются так называемые «внеядерные» файловые системы W:[FUSE] (Filesystem in USErspace), реализуемые не ядерными модулями файловых систем (как ext4, nfs, rgos и пр.), а обычными программами, запущенными в обычных процессах и работающими вне ядра.

В примере из листинга 3.26 показано, как можно без распаковки смонтировать сжатый архив исходных текстов ядра linux-4.2.3.tar.xz и прочитать отдельный файл fuse.txt.

Листинг 3.26 Внеядерная файловая система “fuse.archivemount”

```
finn@ubuntu:~$ wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.3.9.tar.xz
...
finn@ubuntu:~$ file linux-5.3.9.tar.xz
linux-5.3.9.tar.xz: XZ compressed data
finn@ubuntu:~$ archivemount linux-5.3.9.tar.xz ~/mnt/archive
              ⏺ ⏺ ⏺ ⏺ ⏺ ⏺ ⏺ ⏺ ⏺ ⏺ ⏺ ⏺
finn@ubuntu:~$ mount
...
← archivemount on /home/finn/mnt/archive type fuse.archivemount (rw,...)
finn@ubuntu:~$ cd ~/mnt/archive
finn@ubuntu:~/mnt/archive$ ls -l
итого 0
drwxrwxr-x 0  root root 0  ноя  6 15:09 linux-5.3.9
finn@ubuntu:~/mnt/archive$ cd linux-5.3.9/Documentation/filesystems
finn@ubuntu:~/mnt/archive/linux-5.3.9/Documentation/filesystems$ less fuse.txt
finn@ubuntu:~/mnt/archive/linux-5.3.9/Documentation/filesystems$ cd -
finn@ubuntu:~ $ fusermount -u ~/mnt/archive
```

В примере из листинга 3.27 показано, как при помощи сетевого протокола SSH можно смонтировать часть дерева каталогов (домашний каталог пользователя `jake`) с удаленного узла `jake@grex.org` в каталог `~/mnt/net` локального дерева каталогов.

Листинг 3.27. Внеядерная файловая система «fuse.sshfs»

```
finn@ubuntu:~$ sshfs jake@grex.org: ~/mnt/net
The authenticity of host 'grex.org (75.61.90.157)' can't be established.
ECDSA key fingerprint is SHA256:pM03Fe6UTyqtqzUMq5SmTmH5tqUuN9WdvLwdpcEJhSU.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
jake@grex.org's password: P@ssw0rd
```

1 2 3 4 5 6 7 8 9 10

```
finn@ubuntu:~$ mount
```

```
jake@grex.org: on /home/flnn/mnt/net type fuse.sshfs (rw,...)
```

```
finn@ubuntu:~$ cd ~/mnt/net
```

```
finn@ubunty:~/mnt/nets$ ls -l
```

ИТОГО 488

-rw-rw-r-- 1 156620 131077 495604 окт. 10 20:28 OPENME.txz

-rw-r--r-- 1 156620 131077 106 okt. 10 20:30 README.gz

```
finn@ubuntu:~/mnt/net$ zless README.gz
```

```
f1nn@ubuntu:~/mnt/net$ cd ~
```

```
finn@ubuntu:~ $ fusermount -u ~/mnt/net
```

В примере из листинга 3.28 показано, как можно смонтировать зашифровываемый каталог /media/flash/secret USB-flash-накопителя (уже смонтированного в каталог /meida/flash при помощи дисковой файловой системы vfat) в каталог ~/mnt/exposed и скопировать туда файлы, подлежащие зашифровыванию.

Теперь при утере накопителя можно не беспокоиться об информации, попавшей третьим лицам.

Листинг 3.28. Внеядерная файловая система “fuse.end”

Директория "/media/flash/secret" не существует. Создать ее? (y,n) y

Директория "/home/finn/mnt/exposed" не существует. Создать ее? (у,н) у

Листинг 3.28. Продолжение

Создание нового зашифрованного раздела.

Выберите одну из следующих букв:

введите "x" для режима эксперта,

введите "r" для режима максимальной секретности,

любой другая буква для выбора стандартного режима.

?> ↵

Выбрана стандартная конфигурация.

Конфигурация завершена. Создана файловая система
со следующими свойствами:

Шифр файловой системы: "ssl/aes", версия 3:0:2

Шифр файла: "pameio/block", версия 4:0:1

Размер ключа: 192 бит

Размер блока: 1024 байт

Каждый файл содержит 8-байтный заголовок с уникальными IV данными.

Файловые имена зашифрованы с использованием режима сцепления вектора инициализации.

File holes passed through to ciphertext.

Листинг 3.28. Продолжение

Введите пароль для доступа к файловой системе.

Запомните пароль, так как в случае утери его будет невозможно восстановить данные. Тем не менее этот пароль можно изменить с помощью утилиты encfsctl.

Новый пароль EncFS: **\$secr6t ↵**

Повторите пароль EncFS: **\$secr6t ↵**

finn@ubuntu:~\$ mount

➔ encfs on /home/finn/mnt/exposed type fuse.encfs (rw,...)

finn@ubuntu:~\$ cp ~/Изображения/IMG_20150801*.jpg ~/mnt/exposed

finn@ubuntu:~\$ ls -l ~/mnt/exposed/

...
-rw-r--r--	1 finn finn 1014408	окт.	10 19:06	IMG_20140801_123522.jpg	

...
-rw-r--r--	1 finn finn 1728838	окт.	10 19:06	IMG_20140801_124215.jpg	

Листинг 3.28. Продолжение

```
finn@ubuntu:~$ ls -l /media/flash/secret
```

```
... ... ... ...  
-rw-rw-r-- 1 finn finn 0 ноя 18 00:57 JE00j8acowSgiyCueyqqAbohACIKSEM4JShALYNc6PF1l0
```

```
... ... ... ...  
-rw-rw-r-- 1 finn finn 0 ноя 18 00:57 znchYI0d4VP1q7u5UkH2wNqzeYhFeLzTpKDN45Sr1o8T1
```

```
finn@ubuntu:~$ file ~/mnt/exposed/IMG_20140801_123522.jpg
```

```
IMG_20140801_123522.jpg: JPEG image data, EXIF standard
```

```
finn@ubuntu:~$ file /media/flash/secret/9WgX2m6hiNNz8,ii7UI1AIPetU3qGuLBNyww9eHTiNNi-: data
```

```
finn@ubuntu:~$ fusermount -u /media/flash/secret
```

В примере из листинга 3.29 показано, как можно монтировать файловые системы fuse друг поверх друга в стек — смонтировать содержимое дерева каталогов FTP- сервера mirror.yandex.ru, далее смонтировать содержимое ISO-образа FreeBSD-12.1- RELEASE-and64-dvdl.iso, а затем смонтировать архив исходных текстов src.txz.

При чтении файла страницы руководства ls.1 файловые системы будут прозрачны и на лету (!) извлекать файл из архива, архив из образа и образ с сервера без предварительных скачиваний и распакований.

Листинг 3.29. Внедерные файловые системы «fuse.curlftpfs», «fuse.fuseiso» и стекирование файловых систем

```
finn@ubuntu:~$ curlftpfs mirror.yandex.ru ~/mnt/net
          ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎
finn@ubuntu:~$ fuseiso ~/mnt/net/freebsd/releases/ISO-IMAGES/12.1/FreeBSD-12.1-RELEASE-amd64-dvd1.iso ~/mnt/cd
          ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎
finn@ubuntu:~$ archivemount ~/mnt/cd/usr/freebsd-dist/src.txz ~/mnt/archive
          ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎
finn@ubuntu:~$ mount
curlftpfs#ftp://mirror.yandex.ru/ on /home/finn/mnt/net type fuse (rw,...)
fuseiso on /home/finn/mnt/cd type fuse.fuseiso (rw,...)
archivemount on /home/finn/mnt/archive type fuse.archivemount (rw,...)

finn@ubuntu:~$ man ~/mnt/archive/usr/src/bin/ls/ls.1
          ...           ...           ...           ...
LS(1)                                BSD General Commands Manual

NAME
     ls - list directory contents
          ...           ...           ...           ...

finn@ubuntu:~$ fusermount -u ~/mnt/archive
finn@ubuntu:~$ fusermount -u ~/mnt/cd
finn@ubuntu:~$ fusermount -u ~/mnt/net
```

Дискреционное разграничение доступа

В Linux, как и в любой многопользовательской системе, абсолютно естественным образом возникает задача разграничения доступа субъектов — пользователей к объектам — файлам дерева каталогов.

Один из подходов к разграничению доступа — так называемый дискреционный (от англ, *discretion* — чье-либо усмотрение) — предполагает назначение владельцев объектов, которые по собственному усмотрению определяют права доступа субъектов (других пользователей) к объектам (файлам), которыми владеют.

Дискреционные механизмы разграничения доступа используются для разграничения прав доступа процессов как обычных пользователей , так и для ограничения прав системных программ (например, служб операционной системы), которые работают от лица псевдопользовательских учетных записей.

В примере из листинга 3.30 при помощи команды `ps` проиллюстрированы процессы операционной системы, выполняющиеся от лица разных учетных записей.

```
finn@ubuntu:~$ ps auxf
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	2	0.0	0.0	0	0	?	S	ноя17	0:00	[kthreadd]
root	3	0.0	0.0	0	0	?	I<	ноя17	0:00	_ [rcu_gp]
root	4	0.0	0.0	0	0	?	I<	ноя17	0:00	_ [rcu_par_gp]
				
root	1	0.0	0.2	167164	10936	?	Ss	ноя17	0:06	/sbin/init splash
systemd+	577	0.0	0.2	20784	9836	?	Ss	ноя17	0:01	/lib/systemd/systemd-resolved
syslog	606	0.0	0.1	224360	4584	?	Ssl	ноя17	0:01	/usr/sbin/rsyslogd -n -iNONE
message+	617	0.0	0.1	9808	6448	?	Ss	ноя17	0:18	/usr/bin/dbus-daemon --system ...
				
avahi	628	0.0	0.0	8532	3648	?	Ss	ноя17	0:00	avahi-daemon: running ...
daemon	675	0.0	0.0	3736	2232	?	Ss	ноя17	0:00	/usr/sbin/atd -f
				
whoopsie	812	0.0	0.3	330936	15844	?	Ssl	ноя17	0:00	/usr/bin/whoopsie -f
root	1145	0.0	0.1	37168	4040	?	Ss	ноя17	0:00	/usr/lib/postfix/sbin/master -w
● postfix	1150	0.0	0.1	37556	5520	?	S	ноя17	0:00	_ qmgr -l -t unix -u
postfix	21924	0.0	0.1	37504	5400	?	S	00:39	0:00	_ pickup -l -t unix -u -c
				
root	2539	0.0	0.2	251392	9792	?	Ssl	ноя17	0:00	/usr/sbin/gdm3
root	17750	0.0	0.2	316608	9932	?	Sl	ноя17	0:00	_ gdm-session-worker [pam/gdm-...]
finn	17764	0.0	0.1	166540	6908	tty3	Ssl+	ноя17	0:00	_ /usr/lib/gdm3/gdm-x- ...
finn	17766	0.1	1.8	237008	75900	tty3	Sl+	ноя17	0:23	_ /usr/lib/xorg/Xorg ...
● finn	17774	0.0	0.3	194680	15940	tty3	Sl+	ноя17	0:00	_ /usr/lib/gnome-sessi... ...
				
finn	2987	0.0	0.3	21112	12356	?	Ss	ноя17	0:03	/lib/systemd/systemd --user
finn	2992	0.0	0.0	168572	3124	?	S	ноя17	0:00	_ (sd-pam)
● finn	17921	0.7	8.2	2641464	331536	?	Ssl	ноя17	2:01	_ /usr/bin/gnome-shell

Листинг 3.29.Субъекты разграничения прав доступа: пользователи и псевдопользователи

Владельцы и режим доступа к файлам

В рамках дискреционного разграничения доступа каждому файлу назначены пользователь-владелец и группа-владелец в файла (листинг 3.31).

Листинг 3.31. Владельцы файлов

```
finn@ubuntu:~$ ls -la /etc/profile .profile
-rw-r--r-- 1 root root 581 авг 27 21:31 /etc/profile
-rw-r--r-- 1 finn      finn 807 ноя 13 00:25 .profile
finn@ubuntu:~$ stat .profile
...
...
...
...
Доступ: (0644/-rw-r--r--) ① Uid: ( 1000/    finn)  ② Gid: ( 1000/    finn)
...
...
...
...
```

Назначаются владельцы файлов при их создании.

По умолчанию пользователем- владельцем файла становится пользователь, создавший файл, а группой-владельцем файла становится его первичная группа (листинг 3.32).

Листинг 3.32. Назначение владельцев файлов при создании

```
finn@ubuntu:~$ id  
uid=1000(finn) gid=1000(finn) группы=1000(finn),4(adm),24(cdrom),...,131(sambashare)  
finn@ubuntu:~$ nano README  
finn@ubuntu:~$ ls -la README  
-rw-rw-r-- 1 finn finn 3 ноя 18 01:20 README
```

Изменить (листинг 3.33) пользователя-владельца файлов может только суперпользователь root при помощи команды chown, а группу-владельца — владелец файла при помощи команды chgrp(1), но только на ту к которой он сам принадлежит.

Листинг 3.33. Смена владельцев файлов

```
finn@ubuntu:~$ ls -la README  
-rw-rw-r-- 1 finn finn 3 ноя 18 01:20 README
```

```
finn@ubuntu:~$ chown jake README
```

- ❶ chown: изменение владельца 'README': Операция не позволена

```
finn@ubuntu:~$ id  
uid=1000(finn) gid=1000(finn) группы=1000(finn),4(adm),24(cdrom),...,131(sambashare)
```

- ❷ finn@ubuntu:~\$ chgrp adm README

```
finn@ubuntu:~$ ls -la README  
-rw-rw-r-- 1 finn adm 3 ноя 18 01:20 README
```

```
finn@ubuntu:~$ chgrp daemon README
```

- ❸ chgrp: изменение группы для 'README': Операция не позволена

Листинг 3.33 Продолжение

```
finn@ubuntu:~$ sudo chown jake README  
[sudo] password for user: <пароль finn'a>  
finn@ubuntu:~$ ls -la README  
-rw-rw-r-- 1 jake admin 3 ноя 18 01:20 README
```

В примере из листинга 3.16 стоит обратить внимание на владельцев специальных файлов устройств виртуальных терминалов — ими назначены пользователи, совершившие вход в систему.

Специальный файл устройства терминалов, на которых вход не осуществлен, принадлежит суперпользователю root, например как в случае с терминалом /dev/tty9.

Такие назначения естественным образом делаются при входе пользователей в систему и для того, чтобы они могли распоряжаться правами устройства терминала, через который осуществили вход (см. листинг 3.41).

Базовые права доступа и дополнительные атрибуты

Для разграничения действий над файлами определены три базовых права доступа (базовые разрешения): чтение г — «read», запись w — «write» и выполнение x — «execute», соответствующие разрешению выполнять системные вызовы read, write и execve.

Каждое базовое право назначается на файл тому или иному пользователю или группе, разрешая соответствующую операцию.

В наследии классической UNIX определены только три субъекта (листинг 3.34), которым назначаются базовые права — пользователь-владелец (owner), группа-владелец (group owner) и все остальные (others) .

Совокупность их базовых прав называется режимом доступа (access mode) к файлу.

Базовое право может быть назначено г, w или x или отозвано -, поэтому в метаданных файла представляется одним битом, а для режима доступа требуется девять бит: по три бита прав на каждый из трех субъектов доступа.

Компактно режим доступа может быть записан соответствующим числом в восьмеричной системе счисления rw- г- - г- - = 110100100 = 644в.

Листинг 3.34 Режим доступа к файлу

```
finn@ubuntu:~$ ls -l .profile  
-rw-r--r-- 1 finn finn 677 ноя 13 00:25 .profile
```

Проверка режима доступа (листинг 3.35) при операциях с файлами проверяется «слева направо» до первого совпадения.

Если пользователь, осуществляющий операцию с файлом, является его владельцем, тогда используются только права владельца.

В противном случае проверяется членство пользователя, осуществляющего операцию с файлом, в группе-владельцев файла, и тогда используются только права группы-владельцев.

В других случаях используются права для всех остальных , а для суперпользователя root вообще никакие проверки не осуществляются.

Листинг 3.35. Использование режима доступа к файлу

```
finn@ubuntu:~$ id finn
uid=1000(finn) gid=1000(finn) группы=1000(finn),4(adm),24(cdrom),...,131(sambashare)
finn@ubuntu:~$ ls -l README.*
❶ -rw-r--r-- 1 finn      adm    2471 окт. 11 01:12 README.finn
--w-r--r-- 1 finn      adm    2471 окт. 11 01:12 README.finn.locked
❷ -rw-r--r-- 1 jake      adm    776 окт. 11 01:12 README.jake
-rw----r-- 1 bubblegum  adm    171 окт. 11 01:12 README.bubblegum
❸ -rw-r--r-- 1 marceline marceline 16 окт. 11 01:12 README.marceline
-rw-r----- 1 iceking   iceking  31 окт. 11 01:12 README.iceking
finn@ubuntu:~$ wc README.*
wc: README.bubblegum: Отказано в доступе
24 177 2471 README.finn
wc: README.finn.locked: Отказано в доступе
wc: README.iceking: Отказано в доступе
12 70 776 README.jake
1 1 16 README.marceline
37 248 3263 итого
```

Режим доступа новых файлов

Назначается режим доступа файлов при их создании программой, создавшей файл, исходя из назначения файла, но с учетом пожеланий (точнее, нежеланий) пользователя.

Так, например, текстовые редакторы назначают создаваемым (текстовым) файлам права `rw` для всех субъектов, а компиляторы назначают создаваемым (программным) файлам права `rwx` для всех субъектов.

Пользователь может выразить свое нежелание назначать вновь создаваемым файлам те или иные права доступа для тех или иных субъектов, установив так называемую реверсивную (т. е. обратную, символизирующую НЕжелание) маску доступа при помощи встроенной команды интерпретатора `umask` (листинг 3.36).

```
finn@ubuntu:~$ umask  
0002  
  
finn@ubuntu:~$ umask -S  
u=rwx,g=rwx,o=rx  
  
finn@ubuntu:~$ touch common.jnl  
finn@ubuntu:~$ ls -l common.jnl  
-rw-rw-r-- 1 finn finn 0 ноя 18 01:37 common.jnl  
  
finn@ubuntu:~$ umask g-w,o-rwx  
  
finn@ubuntu:~$ umask  
0027  
  
finn@ubuntu:~$ umask -S  
u=rwx,g=rx,o=  
  
finn@ubuntu:~$ touch group.jnl  
finn@ubuntu:~$ ls -l group.jnl  
-rw-r----- 1 finn finn 0 ноя 18 01:37 group.jnl  
  
finn@ubuntu:~$ umask g=  
  
finn@ubuntu:~$ umask  
0077  
  
finn@ubuntu:~$ umask -S  
u=rwx,g=,o=  
  
finn@ubuntu:~$ touch private.jnl  
finn@ubuntu:~$ ls -l private.jnl  
-rw----- 1 finn finn 0 ноя 18 01:38 private.jnl
```

Листинг 3.36. Реверсивная маска доступа

Изменять режима доступа разрешено непосредственному пользователю — владельцу файла, но не членами группы-владельцев, что иллюстрирует листинг 3.37 при помощи команды chmod.

```
finn@ubuntu:~$ id finn
uid=1000(finn) gid=1000(finn) группы=1000(finn),4(admin),24(cdrom),...,131(sambashare)
finn@ubuntu:~$ ls -l README.*
-rw-r--r-- 1 finn      admin    2471 окт. 11 01:12 README.finn
-rw-r--r-- 1 jake      admin    776  окт. 11 01:12 README.jake
...
...
finn@ubuntu:~$ chmod g-r,o-r README.finn
finn@ubuntu:~$ ls -la README.finn
-rw----- 1 finn      admin    2471 окт. 11 01:12 README.finn
finn@ubuntu:~$ chmod g-r,o-r README.jake
chmod: изменение прав доступа для 'README.jake': Операция не позволена
```

Семантика режима доступа разных типов файл

Права доступа `g`, `w`, `x` для обычных файлов представляются чем-то интуитивно понятным, но для других типов файлов это не совсем так. Например, каталог (см. рис. 3.2) содержит список имен файлов, поэтому право `w` для каталога — это право записи в этот список и право стирания из этого списка, что трансформируется в право удаления файлов из каталога и создания файлов в каталоге.

Аналогично, право `g` для каталога — это право просмотра списка имен его файлов. И наконец, право `x` для каталога является правом прохода в каталог, т. е. позволяет обращаться к файлам внутри каталога по их имени (листинг 3.38).

Листинг 3.38. Права доступа к каталогу

```
finn@ubuntu:~$ mkdir folder
finn@ubuntu:~$ ls -lad folder/
drwxrwxr-x 2 finn finn 4096 окт. 12 09:37 folder/
finn@ubuntu:~$ cp /etc/magic folder
finn@ubuntu:~$ chmod u-w folder
finn@ubuntu:~$ ls -lad folder/
dr-xrwxr-x 2 finn finn 4096 окт. 12 09:40 folder/
finn@ubuntu:~$ cp /etc/localtime folder/
cp: невозможно создать обычный файл «folder/localtime»: Отказано в доступе
finn@ubuntu:~$ rm folder/magic
rm: невозможно удалить «folder/magic»: Отказано в доступе
finn@ubuntu:~$ ls -li folder/
итого 4
20318203 -rw-r--r-- 1 finn finn 111 окт. 12 09:40 magic
```

Листинг 3.38. Продолжение

```
finn@ubuntu:~$ chmod u-r folder
finn@ubuntu:~$ ls -lad folder/
d--xrwxr-x 2 finn finn 4096 окт. 12 00:40 folder/
finn@ubuntu:~$ ls -li folder/
ls: невозможно открыть каталог folder/: Отказано в доступе
finn@ubuntu:~$ ls -li folder/magic
! 20318203 -rwxr--r-- 1 finn finn 111 окт. 12 00:40 folder/magic
finn@ubuntu:~$ chmod u-x folder/
```

Листинг 3.38. Продолжение

```
finn@ubuntu:~$ ls -lad folder/
d---rwxr-x 2 finn finn 4096 окт. 12 00:40 folder/
finn@ubuntu:~$ ls -li folder/magic
ls: невозможно получить доступ к folder/magic: Отказано в доступе
finn@ubuntu:~$ chmod u+rw folder/
finn@ubuntu:~$ ls -lad folder/
drw-rwxr-x 2 finn finn 4096 окт. 12 00:40 folder/
finn@ubuntu:~$ cp /etc/localtime folder/
cp: не удалось выполнить stat для «folder/localtime»: Отказано в доступе
finn@ubuntu:~$ rm folder/magic
rm: невозможно удалить «folder/magic»: Отказано в доступе
finn@ubuntu:~$ ls -l folder/
ls: невозможно получить доступ к folder/magic: Отказано в доступе
итого 0
-?????????? ? ? ? ?
? magic
```

Листинг 3.38. Продолжение

```
finn@ubuntu:~$ chmod u=rwx folder/
finn@ubuntu:~$ ls -ld folder/
drwxrwxr-x 2 finn finn 4096 окт. 12 00:40 folder/
finn@ubuntu:~$ cd folder/
finn@ubuntu:~/folder$ ls -l
итого 4
-rw-r--r-- 1 finn finn 111 окт. 12 00:40 magic
finn@ubuntu:~/folder$ chmod a= magic
finn@ubuntu:~/folder$ ls -l magic
-r----- 1 finn finn 111 окт. 12 00:40 magic
finn@ubuntu:~/folder$ rm magic
! rm: удалить защищенный от записи обычный файл «magic»? у
finn@ubuntu:~/folder$ ls -l
! итого 0
```

Для жестких ссылок права доступа не существуют вовсе —; они просто являются теми же правами, что и права целевого файла, в силу того что права доступа хранятся в метаданных.

Для символических ссылок семантика прав сохранена такой же, как и у жестких ссылок, с тем лишь различием, что права символических ссылок существуют отдельно от целевых файлов, но никогда не проверяются (см. `symlink`).

Для изменения прав доступа самих символических ссылок даже не существует специальной команды — при использовании `chmod(l)` со ссылкой всегда будут изменяться права целевого файла (листинг 3.39).

Листинг 339. Права доступа ссылок

```
finn@ubuntu:~$ ls -l README.finn
-rwxr--r-- 1 finn finn 2471 окт. 11 01:13 README.finn
finn@ubuntu:~$ ln README.finn read.me
finn@ubuntu:~$ ln -s README.finn readme.1st
finn@ubuntu:~$ ls -l README.finn read.me readme.1st
-rwxr--r-- 2 finn finn 2471 окт. 11 01:13 read.me
lrwxrwxrwx 1 finn finn 11 окт. 12 01:19 readme.1st -> README.finn
-rwxr--r-- 2 finn finn 2471 окт. 11 01:13 README.finn
finn@ubuntu:~$ chmod g+w read.me
finn@ubuntu:~$ ls -l README.finn read.me readme.1st
-rwxrw-r-- 2 finn finn 2471 окт. 11 01:13 read.me
lrwxrwxrwx 1 finn finn 11 окт. 12 01:19 readme.1st -> README.finn
-rwxrw-r-- 2 finn finn 2471 окт. 11 01:13 README.finn
finn@ubuntu:~$ chmod o-r readme.1st
finn@ubuntu:~$ ls -l README.finn read.me readme.1st
-rwxrw---- 2 finn finn 2471 окт. 11 01:13 read.me
lrwxrwxrwx 1 finn finn 11 окт. 12 01:19 readme.1st -> README.finn
-rwxrw---- 2 finn finn 2471 окт. 11 01:13 README.finn
```

Для специальных файлов устройств, именованных каналов и сокетов право x не определено, а права g и w стоит воспринимать как права ввода и вывода информации на устройство и как права передачи и приема информации через средство взаимодействия.

Специальные файловые системы. Внеядерные
файловые системы. Дисcretionное разграничение
доступа. Семантика режима доступа разных типов
файлов. Режим доступа новых файлов.

Лекция 4+

Специальные файловые системы

Развитие идеи файла как единицы обеспечения доступа к информации привело к тому, что абстракцию файловой системы перенесли и на другие сущности, доступ к которым стал организовываться в виде иерархии файлов.

Например, информацию о процессах, нитях и прочих сущностях ядра операционной системы и используемых ими ресурсах предоставляет программам виде файлов (!) псевдофайловая система `/proc`.

Таким же образом, информацию об аппаратных устройствах, обнаруженных ядром операционной системы на шинах PCI, USB, SCSI и пр., предоставляет псевдофайловая система `/sysfs`.

Различные утилиты, пользующиеся ядерной информацией, например показывающие нагрузку на операционную систему `uptime` или списки процессов и загруженных модулей (драйверов) ядра операционной системы — `ps` и `lsmod`, пользуются псевдофайловой системой `/proc`, в чем позволяет убедиться трассировка системных вызовов `open` (листинг 3.24).

Листинг 3.24. Псевдофайловая система

```
finn@ubuntu:~$ strace -fe open,openat uptime
...
...
...
openat(AT_FDCWD, "/proc/sys/kernel/osrelease", O_RDONLY) = 3
...
...
openat(AT_FDCWD, "/proc/uptime", O_RDONLY) = 3
...
...
openat(AT_FDCWD, "/proc/loadavg", O_RDONLY) = 4
...
...
15:42:32 up 12 days, 5:27, 7 users, load average: 0.48 ▶, 0.33, 0.32
finn@ubuntu:~$ cat /proc/uptime
1056774.23 1667210.55
finn@ubuntu:~$ cat /proc/loadavg
0.48 ▶ 0.33 0.32 1/623 14277
```

Аналогично, утилиты, показывающие список устройств на шинах PCI, USB и SCSI — `lspci`, `lsusb` и `lsscsi`, пользуются псевдофайловой системой `sysfs` (листинг 3.25).

Листинг 3.25 Псевдофайловая система `sysfs`

```
finn@ubuntu:~$ strace -fe open,openat lspci -nn
...
openat(AT_FDCWD, "/sys/bus/pci/devices/0000:00:02.0/resource", O_RDONLY) = 4
openat(AT_FDCWD, "/sys/bus/pci/devices/0000:00:02.0/irq", O_RDONLY) = 4
openat(AT_FDCWD, "/sys/bus/pci/devices/0000:00:02.0/vendor", O_RDONLY) = 4
openat(AT_FDCWD, "/sys/bus/pci/devices/0000:00:02.0/device", O_RDONLY) = 4
openat(AT_FDCWD, "/sys/bus/pci/devices/0000:00:02.0/class", O_RDONLY) = 4
openat(AT_FDCWD, "/sys/bus/pci/devices/0000:00:02.0/config", O_RDONLY) = 3
...
00:02.0 VGA compatible controller [0300 ↵]: Intel Corporation 2nd Generation Core Processor
Family Integrated Graphics Controller [8086 ↵:0116 ↵] (rev 09)
finn@ubuntu:~$ cat /sys/bus/pci/devices/0000:00:02.0/vendor
0x8086 ↵
finn@ubuntu:~$ cat /sys/bus/pci/devices/0000:00:02.0/device
0x0116 ↵
finn@ubuntu:~$ cat /sys/bus/pci/devices/0000:00:02.0/class
0x030000 ↵
```

Внеядерные файловые системы

Существование разных по своей сути файловых систем в едином дереве файлов и каталогов позволяет использовать для работы с его файлами «обычные» программы, ничего не знающие ни о местоположении, ни о свойствах самих источников информации этих файлов.

Таким же «файловым» образом можно организовать доступ к файлам внутри GZ/ZIP/...-архивов, файлам внутри ISO-образов CD/DVD-дисков, файлам зашифрованных каталогов, файлам других узлов сети, медиафайлам на медиаустройствах (плееры, смартфоны) и т. п.

Файловая абстракция в определенных случаях оказывается удобной даже для осуществления доступа к информации, организованной совершенно «нефайловым» образом — к данным таблиц и представлений внутри реляционных баз данных, к элементам и атрибутам внутри XML-файлов и пр.

Для реализации такого «файлового» доступа к произвольным источниками информации используются так называемые «внеядерные» файловые системы W:[FUSE] (Filesystem in USErspace), реализуемые не ядерными модулями файловых систем (как ext4, nfs, rgos и пр.), а обычными программами, запущенными в обычных процессах и работающими вне ядра.

В примере из листинга 3.26 показано, как можно без распаковки смонтировать сжатый архив исходных текстов ядра linux-4.2.3.tar.xz и прочитать отдельный файл fuse.txt.

Листинг 3.26 Внеядерная файловая система “fuse.archivemount”

```
finn@ubuntu:~$ wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.3.9.tar.xz
...
finn@ubuntu:~$ file linux-5.3.9.tar.xz
linux-5.3.9.tar.xz: XZ compressed data
finn@ubuntu:~$ archivemount linux-5.3.9.tar.xz ~/mnt/archive
              ⏺ ⏺ ⏺ ⏺ ⏺ ⏺ ⏺ ⏺ ⏺ ⏺ ⏺ ⏺
finn@ubuntu:~$ mount
...
← archivemount on /home/finn/mnt/archive type fuse.archivemount (rw,...)
finn@ubuntu:~$ cd ~/mnt/archive
finn@ubuntu:~/mnt/archive$ ls -l
итого 0
drwxrwxr-x 0  root root 0  ноя  6 15:09 linux-5.3.9
finn@ubuntu:~/mnt/archive$ cd linux-5.3.9/Documentation/filesystems
finn@ubuntu:~/mnt/archive/linux-5.3.9/Documentation/filesystems$ less fuse.txt
finn@ubuntu:~/mnt/archive/linux-5.3.9/Documentation/filesystems$ cd -
finn@ubuntu:~ $ fusermount -u ~/mnt/archive
```

В примере из листинга 3.27 показано, как при помощи сетевого протокола SSH можно смонтировать часть дерева каталогов (домашний каталог пользователя `jake`) с удаленного узла `jake@grex.org` в каталог `~/mnt/net` локального дерева каталогов.

Листинг 3.27. Внеядерная файловая система «fuse.sshfs»

В примере из листинга 3.28 показано, как можно смонтировать зашифровываемый каталог /media/flash/secret USB-flash-накопителя (уже смонтированного в каталог /meida/flash при помощи дисковой файловой системы vfat) в каталог ~/mnt/exposed и скопировать туда файлы, подлежащие зашифровыванию.

Теперь при утере накопителя можно не беспокоиться об информации, попавшей третьим лицам.

Листинг 3.28. Внеядерная файловая система “fuse.end”

Директория "/media/flash/secret" не существует. Создать ее? (y,n) y

Директория "/home/finn/mnt/exposed" не существует. Создать ее? (у,н) у

Листинг 3.28. Продолжение

Создание нового зашифрованного раздела.

Выберите одну из следующих букв:

введите "x" для режима эксперта,

введите "r" для режима максимальной секретности,

любой другая буква для выбора стандартного режима.

?> ←

Выбрана стандартная конфигурация.

Конфигурация завершена. Создана файловая система
со следующими свойствами:

Шифр файловой системы: "ssl/aes", версия 3:0:2

Шифр файла: "pameio/block", версия 4:0:1

Размер ключа: 192 бит

Размер блока: 1024 байт

Каждый файл содержит 8-байтный заголовок с уникальными IV данными.

Файловые имена зашифрованы с использованием режима сцепления вектора инициализации.

File holes passed through to ciphertext.

Листинг 3.28. Продолжение

Введите пароль для доступа к файловой системе.

Запомните пароль, так как в случае утери его будет невозможно восстановить данные. Тем не менее этот пароль можно изменить с помощью утилиты encfsctl.

Новый пароль EncFS: **\$secr6t ↵**

Повторите пароль EncFS: **\$secr6t ↵**

finn@ubuntu:~\$ mount

➔ encfs on /home/finn/mnt/exposed type fuse.encfs (rw,...)

finn@ubuntu:~\$ cp ~/Изображения/IMG_20150801*.jpg ~/mnt/exposed

finn@ubuntu:~\$ ls -l ~/mnt/exposed/

...
-rw-r--r--	1 finn finn 1014408	окт.	10 19:06	IMG_20140801_123522.jpg	

...
-rw-r--r--	1 finn finn 1728838	окт.	10 19:06	IMG_20140801_124215.jpg	

Листинг 3.28. Продолжение

```
finn@ubuntu:~$ ls -l /media/flash/secret
```

```
... ... ... ...  
-rw-rw-r-- 1 finn finn 0 ноя 18 00:57 JE00j8acowSgiyCueyqqAbohACIKSEM4JShALYNc6PF1l0
```

```
... ... ... ...  
-rw-rw-r-- 1 finn finn 0 ноя 18 00:57 znchYI0d4VP1q7u5UkH2wNqzeYhFeLzTpKDN45Sr1o8T1
```

```
finn@ubuntu:~$ file ~/mnt/exposed/IMG_20140801_123522.jpg
```

```
IMG_20140801_123522.jpg: JPEG image data, EXIF standard
```

```
finn@ubuntu:~$ file /media/flash/secret/9WgX2m6hiNNz8,ii7UI1AIPetU3qGuLBNyww9eHTiNNi-: data
```

```
finn@ubuntu:~$ fusermount -u /media/flash/secret
```

В примере из листинга 3.29 показано, как можно монтировать файловые системы fuse друг поверх друга в стек — смонтировать содержимое дерева каталогов FTP- сервера mirror.yandex.ru, далее смонтировать содержимое ISO-образа FreeBSD-12.1- RELEASE-and64-dvdl.iso, а затем смонтировать архив исходных текстов src.txz.

При чтении файла страницы руководства ls.1 файловые системы будут прозрачны и на лету (!) извлекать файл из архива, архив из образа и образ с сервера без предварительных скачиваний и распакований.

Листинг 3.29. Внедерные файловые системы «fuse.curlftpfs», «fuse.fuseiso» и стекирование файловых систем

```
finn@ubuntu:~$ curlftpfs mirror.yandex.ru ~/mnt/net
                                          ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎
finn@ubuntu:~$ fuseiso ~/mnt/net/freebsd/releases/ISO-IMAGES/12.1/FreeBSD-12.1-RELEASE-amd64-dvd1.iso ~/mnt/cd
                                          ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎
finn@ubuntu:~$ archivemount ~/mnt/cd/usr/freebsd-dist/src.txz ~/mnt/archive
                                          ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎ ⏎
finn@ubuntu:~$ mount
curlftpfs#ftp://mirror.yandex.ru/ on /home/finn/mnt/net type fuse (rw,...)
fuseiso on /home/finn/mnt/cd type fuse.fuseiso (rw,...)
archivemount on /home/finn/mnt/archive type fuse.archivemount (rw,...)

finn@ubuntu:~$ man ~/mnt/archive/usr/src/btn/ls/ls.1
          ...           ...           ...           ...
LS(1)                                BSD General Commands Manual

NAME
     ls - list directory contents
          ...           ...           ...           ...

finn@ubuntu:~$ fusermount -u ~/mnt/archive
finn@ubuntu:~$ fusermount -u ~/mnt/cd
finn@ubuntu:~$ fusermount -u ~/mnt/net
```

Дискреционное разграничение доступа

В Linux, как и в любой многопользовательской системе, абсолютно естественным образом возникает задача разграничения доступа субъектов — пользователей к объектам — файлам дерева каталогов.

Один из подходов к разграничению доступа — так называемый дискреционный (от англ, *discretion* — чье-либо усмотрение) — предполагает назначение владельцев объектов, которые по собственному усмотрению определяют права доступа субъектов (других пользователей) к объектам (файлам), которыми владеют.

Дискреционные механизмы разграничения доступа используются для разграничения прав доступа процессов как обычных пользователей , так и для ограничения прав системных программ (например, служб операционной системы), которые работают от лица псевдопользовательских учетных записей.

В примере из листинга 3.30 при помощи команды `ps` проиллюстрированы процессы операционной системы, выполняющиеся от лица разных учетных записей.

```
finn@ubuntu:~$ ps auxfu
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	2	0.0	0.0	0	0	?	S	ноя17	0:00	[kthreadd]
root	3	0.0	0.0	0	0	?	I<	ноя17	0:00	_ [rcu_gp]
root	4	0.0	0.0	0	0	?	I<	ноя17	0:00	_ [rcu_par_gp]
			
root	1	0.0	0.2	167164	10936	?	Ss	ноя17	0:06	/sbin/init splash
systemd+	577	0.0	0.2	20784	9836	?	Ss	ноя17	0:01	/lib/systemd/systemd-resolved
syslog	606	0.0	0.1	224360	4584	?	Ssl	ноя17	0:01	/usr/sbin/rsyslogd -n -iNONE
message+	617	0.0	0.1	9808	6448	?	Ss	ноя17	0:18	/usr/bin/dbus-daemon --system ...
			
avahi	628	0.0	0.0	8532	3648	?	Ss	ноя17	0:00	avahi-daemon: running ...
daemon	675	0.0	0.0	3736	2232	?	Ss	ноя17	0:00	/usr/sbin/atd -f
			
whoopsie	812	0.0	0.3	330936	15844	?	Ssl	ноя17	0:00	/usr/bin/whoopsie -f
root	1145	0.0	0.1	37168	4040	?	Ss	ноя17	0:00	/usr/lib/postfix/sbin/master -w
● postfix	1150	0.0	0.1	37556	5520	?	S	ноя17	0:00	_ qmgr -l -t unix -u
postfix	21924	0.0	0.1	37504	5400	?	S	00:39	0:00	_ pickup -l -t unix -u -c
			
root	2539	0.0	0.2	251392	9792	?	Ssl	ноя17	0:00	/usr/sbin/gdm3
root	17750	0.0	0.2	316608	9932	?	Sl	ноя17	0:00	_ gdm-session-worker [pam/gdm-...]
finn	17764	0.0	0.1	166540	6908	tty3	Ssl+	ноя17	0:00	_ /usr/lib/gdm3/gdm-x- ...
finn	17766	0.1	1.8	237008	75900	tty3	Sl+	ноя17	0:23	_ /usr/lib/xorg/Xorg ...
● finn	17774	0.0	0.3	194680	15940	tty3	Sl+	ноя17	0:00	_ /usr/lib/gnome-sessi...
			
finn	2987	0.0	0.3	21112	12356	?	Ss	ноя17	0:03	/lib/systemd/systemd --user
finn	2992	0.0	0.0	168572	3124	?	S	ноя17	0:00	_ (sd-pam)
● finn	17921	0.7	8.2	2641464	331536	?	Ssl	ноя17	2:01	_ /usr/bin/gnome-shell

Листинг 3.29.Субъекты разграничения прав доступа: пользователи и псевдопользователи

Владельцы и режим доступа к файлам

В рамках дискреционного разграничения доступа каждому файлу назначены пользователь-владелец и группа-владелец в файла (листинг 3.31).

Листинг 3.31. Владельцы файлов

```
finn@ubuntu:~$ ls -la /etc/profile .profile
-rw-r--r-- 1 root root 581 авг 27 21:31 /etc/profile
-rw-r--r-- 1 finn      finn 807 ноя 13 00:25 .profile
finn@ubuntu:~$ stat .profile
...
...
...
Доступ: (0644/-rw-r--r--) ① Uid: ( 1000/    finn)  ② Gid: ( 1000/    finn)
...
...
...
```

Назначаются владельцы файлов при их создании.

По умолчанию пользователем- владельцем файла становится пользователь, создавший файл, а группой-владельцем файла становится его первичная группа (листинг 3.32).

Листинг 3.32. Назначение владельцев файлов при создании

```
flinn@ubuntu:~$ id  
uid=1000(flinn) gid=1000(flinn) группы=1000(flinn),4(adm),24(cdrom),...,131(sambashare)  
flinn@ubuntu:~$ nano README  
flinn@ubuntu:~$ ls -la README  
-rw-rw-r-- 1 flinn flinn 3 ноя 18 01:20 README
```

Изменить (листинг 3.33) пользователя-владельца файлов может только суперпользователь root при помощи команды chown, а группу-владельца — владелец файла при помощи команды chgrp(1), но только на ту к которой он сам принадлежит.

Листинг 3.33. Смена владельцев файлов

```
finn@ubuntu:~$ ls -la README  
-rw-rw-r-- 1 finn finn 3 ноя 18 01:20 README
```

```
finn@ubuntu:~$ chown jake README
```

- ❶ chown: изменение владельца 'README': Операция не позволена

```
finn@ubuntu:~$ id  
uid=1000(finn) gid=1000(finn) группы=1000(finn),4(adm),24(cdrom),...,131(sambashare)
```

- ❷ finn@ubuntu:~\$ chgrp adm README

```
finn@ubuntu:~$ ls -la README  
-rw-rw-r-- 1 finn adm 3 ноя 18 01:20 README
```

```
finn@ubuntu:~$ chgrp daemon README
```

- ❸ chgrp: изменение группы для 'README': Операция не позволена

Листинг 3.33 Продолжение

```
finn@ubuntu:~$ sudo chown jake README  
[sudo] password for user: <пароль finn'a>  
finn@ubuntu:~$ ls -la README  
-rw-rw-r-- 1 jake admin 3 ноя 18 01:20 README
```

В примере из листинга 3.16 стоит обратить внимание на владельцев специальных файлов устройств виртуальных терминалов — ими назначены пользователи, совершившие вход в систему.

Специальный файл устройства терминалов, на которых вход не осуществлен, принадлежит суперпользователю root, например как в случае с терминалом /dev/tty9.

Такие назначения естественным образом делаются при входе пользователей в систему и для того, чтобы они могли распоряжаться правами устройства терминала, через который осуществили вход (см. листинг 3.41).

Базовые права доступа и дополнительные атрибуты

Для разграничения действий над файлами определены три базовых права доступа (базовые разрешения): чтение г — «read», запись w — «write» и выполнение x — «execute», соответствующие разрешению выполнять системные вызовы read, write и execve.

Каждое базовое право назначается на файл тому или иному пользователю или группе, разрешая соответствующую операцию.

В наследии классической UNIX определены только три субъекта (листинг 3.34), которым назначаются базовые права — пользователь-владелец (owner), группа-владелец (group owner) и все остальные (others) .

Совокупность их базовых прав называется режимом доступа (access mode) к файлу.

Базовое право может быть назначено г, w или x или отозвано -, поэтому в метаданных файла представляется одним битом, а для режима доступа требуется девять бит: по три бита прав на каждый из трех субъектов доступа.

Компактно режим доступа может быть записан соответствующим числом в восьмеричной системе счисления rw- г- - г- - = 110100100 = 644в.

Листинг 3.34 Режим доступа к файлу

```
finn@ubuntu:~$ ls -l .profile  
-rw-r--r-- 1 finn finn 677 ноя 13 00:25 .profile
```

Проверка режима доступа (листинг 3.35) при операциях с файлами проверяется «слева направо» до первого совпадения.

Если пользователь, осуществляющий операцию с файлом, является его владельцем, тогда используются только права владельца.

В противном случае проверяется членство пользователя, осуществляющего операцию с файлом, в группе-владельцев файла, и тогда используются только права группы-владельцев.

В других случаях используются права для всех остальных , а для суперпользователя root вообще никакие проверки не осуществляются.

Листинг 3.35. Использование режима доступа к файлу

```
finn@ubuntu:~$ id finn
uid=1000(finn) gid=1000(finn) группы=1000(finn),4(adm),24(cdrom),...,131(sambashare)
finn@ubuntu:~$ ls -l README.*
❶ -rw-r--r-- 1 finn      adm    2471 окт. 11 01:12 README.finn
--w-r--r-- 1 finn      adm    2471 окт. 11 01:12 README.finn.locked
❷ -rw-r--r-- 1 jake      adm    776 окт. 11 01:12 README.jake
-rw----r-- 1 bubblegum adm    171 окт. 11 01:12 README.bubblegum
❸ -rw-r--r-- 1 marceline marceline 16 окт. 11 01:12 README.marceline
-rw-r----- 1 iceking   iceking  31 окт. 11 01:12 README.iceking
finn@ubuntu:~$ wc README.*
wc: README.bubblegum: Отказано в доступе
24 177 2471 README.finn
wc: README.finn.locked: Отказано в доступе
wc: README.iceking: Отказано в доступе
12 70 776 README.jake
1 1 16 README.marceline
37 248 3263 итого
```

Режим доступа новых файлов

Назначается режим доступа файлов при их создании программой, создавшей файл, исходя из назначения файла, но с учетом пожеланий (точнее, нежеланий) пользователя.

Так, например, текстовые редакторы назначают создаваемым (текстовым) файлам права `rw` для всех субъектов, а компиляторы назначают создаваемым (программным) файлам права `rwx` для всех субъектов.

Пользователь может выразить свое нежелание назначать вновь создаваемым файлам те или иные права доступа для тех или иных субъектов, установив так называемую реверсивную (т. е. обратную, символизирующую НЕжелание) маску доступа при помощи встроенной команды интерпретатора `umask` (листинг 3.36).

```
finn@ubuntu:~$ umask  
0002  
  
finn@ubuntu:~$ umask -S  
u=rwx,g=rwx,o=rx  
  
finn@ubuntu:~$ touch common.jnl  
finn@ubuntu:~$ ls -l common.jnl  
-rw-rw-r-- 1 finn finn 0 ноя 18 01:37 common.jnl  
  
finn@ubuntu:~$ umask g-w,o-rwx  
  
finn@ubuntu:~$ umask  
0027  
  
finn@ubuntu:~$ umask -S  
u=rwx,g=rx,o=  
  
finn@ubuntu:~$ touch group.jnl  
finn@ubuntu:~$ ls -l group.jnl  
-rw-r----- 1 finn finn 0 ноя 18 01:37 group.jnl  
  
finn@ubuntu:~$ umask g=  
  
finn@ubuntu:~$ umask  
0077  
  
finn@ubuntu:~$ umask -S  
u=rwx,g=,o=  
  
finn@ubuntu:~$ touch private.jnl  
finn@ubuntu:~$ ls -l private.jnl  
-rw----- 1 finn finn 0 ноя 18 01:38 private.jnl
```

Листинг 3.36. Реверсивная маска доступа

Изменять режима доступа разрешено непосредственному пользователю — владельцу файла, но не членами группы-владельцев, что иллюстрирует листинг 3.37 при помощи команды chmod.

```
finn@ubuntu:~$ id finn
uid=1000(finn) gid=1000(finn) группы=1000(finn),4(admin),24(cdrom),...,131(sambashare)
finn@ubuntu:~$ ls -l README.*
-rw-r--r-- 1 finn      admin    2471 окт. 11 01:12 README.finn
-rw-r--r-- 1 jake      admin    776  окт. 11 01:12 README.jake
...
...
finn@ubuntu:~$ chmod g-r,o-r README.finn
finn@ubuntu:~$ ls -la README.finn
-rw----- 1 finn      admin    2471 окт. 11 01:12 README.finn
finn@ubuntu:~$ chmod g-r,o-r README.jake
chmod: изменение прав доступа для 'README.jake': Операция не позволена
```

Семантика режима доступа разных типов файл

Права доступа `g`, `w`, `x` для обычных файлов представляются чем-то интуитивно понятным, но для других типов файлов это не совсем так. Например, каталог (см. рис. 3.2) содержит список имен файлов, поэтому право `w` для каталога — это право записи в этот список и право стирания из этого списка, что трансформируется в право удаления файлов из каталога и создания файлов в каталоге.

Аналогично, право `g` для каталога — это право просмотра списка имен его файлов. И наконец, право `x` для каталога является правом прохода в каталог, т. е. позволяет обращаться к файлам внутри каталога по их имени (листинг 3.38).

Листинг 3.38. Права доступа к каталогу

```
finn@ubuntu:~$ mkdir folder
finn@ubuntu:~$ ls -lad folder/
drwxrwxr-x 2 finn finn 4096 окт. 12 09:37 folder/
finn@ubuntu:~$ cp /etc/magic folder
finn@ubuntu:~$ chmod u-w folder
finn@ubuntu:~$ ls -lad folder/
dr-xrwxr-x 2 finn finn 4096 окт. 12 09:40 folder/
finn@ubuntu:~$ cp /etc/localtime folder/
cp: невозможно создать обычный файл «folder/localtime»: Отказано в доступе
finn@ubuntu:~$ rm folder/magic
rm: невозможно удалить «folder/magic»: Отказано в доступе
finn@ubuntu:~$ ls -li folder/
итого 4
20318203 -rw-r--r-- 1 finn finn 111 окт. 12 09:40 magic
```

Листинг 3.38. Продолжение

```
finn@ubuntu:~$ chmod u-r folder
finn@ubuntu:~$ ls -lad folder/
d--xrwxr-x 2 finn finn 4096 окт. 12 00:40 folder/
finn@ubuntu:~$ ls -li folder/
ls: невозможно открыть каталог folder/: Отказано в доступе
finn@ubuntu:~$ ls -li folder/magic
! 20318203 -rwxr--r-- 1 finn finn 111 окт. 12 00:40 folder/magic
finn@ubuntu:~$ chmod u-x folder/
```

Листинг 3.38. Продолжение

```
finn@ubuntu:~$ ls -lad folder/
d---rwxr-x 2 finn finn 4096 окт. 12 00:40 folder/
finn@ubuntu:~$ ls -li folder/magic
ls: невозможно получить доступ к folder/magic: Отказано в доступе
finn@ubuntu:~$ chmod u+rw folder/
finn@ubuntu:~$ ls -lad folder/
drw-rwxr-x 2 finn finn 4096 окт. 12 00:40 folder/
finn@ubuntu:~$ cp /etc/localtime folder/
cp: не удалось выполнить stat для «folder/localtime»: Отказано в доступе
finn@ubuntu:~$ rm folder/magic
rm: невозможно удалить «folder/magic»: Отказано в доступе
finn@ubuntu:~$ ls -l folder/
ls: невозможно получить доступ к folder/magic: Отказано в доступе
итого 0
-?????????? ? ? ? ?
? magic
```

Листинг 3.38. Продолжение

```
finn@ubuntu:~$ chmod u=rwx folder/
finn@ubuntu:~$ ls -ld folder/
drwxrwxr-x 2 finn finn 4096 окт. 12 00:40 folder/
finn@ubuntu:~$ cd folder/
finn@ubuntu:~/folder$ ls -l
итого 4
-rw-r--r-- 1 finn finn 111 окт. 12 00:40 magic
finn@ubuntu:~/folder$ chmod a= magic
finn@ubuntu:~/folder$ ls -l magic
-r----- 1 finn finn 111 окт. 12 00:40 magic
finn@ubuntu:~/folder$ rm magic
! rm: удалить защищенный от записи обычный файл «magic»? у
finn@ubuntu:~/folder$ ls -l
! итого 0
```

Для жестких ссылок права доступа не существуют вовсе —; они просто являются теми же правами, что и права целевого файла, в силу того что права доступа хранятся в метаданных.

Для символических ссылок семантика прав сохранена такой же, как и у жестких ссылок, с тем лишь различием, что права символических ссылок существуют отдельно от целевых файлов, но никогда не проверяются (см. `symlink`).

Для изменения прав доступа самих символических ссылок даже не существует специальной команды — при использовании `chmod(l)` со ссылкой всегда будут изменяться права целевого файла (листинг 3.39).

Листинг 3.39. Права доступа ссылок

```
finn@ubuntu:~$ ls -l README.finn
-rwxr--r-- 1 finn finn 2471 окт. 11 01:13 README.finn
finn@ubuntu:~$ ln README.finn read.me
finn@ubuntu:~$ ln -s README.finn readme.1st
finn@ubuntu:~$ ls -l README.finn read.me readme.1st
-rwxr--r-- 2 finn finn 2471 окт. 11 01:13 read.me
lrwxrwxrwx 1 finn finn 11 окт. 12 01:19 readme.1st -> README.finn
-rwxr--r-- 2 finn finn 2471 окт. 11 01:13 README.finn
finn@ubuntu:~$ chmod g+w read.me
finn@ubuntu:~$ ls -l README.finn read.me readme.1st
-rwxrw-r-- 2 finn finn 2471 окт. 11 01:13 read.me
lrwxrwxrwx 1 finn finn 11 окт. 12 01:19 readme.1st -> README.finn
-rwxrw-r-- 2 finn finn 2471 окт. 11 01:13 README.finn
finn@ubuntu:~$ chmod o-r readme.1st
finn@ubuntu:~$ ls -l README.finn read.me readme.1st
-rwxrw---- 2 finn finn 2471 окт. 11 01:13 read.me
lrwxrwxrwx 1 finn finn 11 окт. 12 01:19 readme.1st -> README.finn
-rwxrw---- 2 finn finn 2471 окт. 11 01:13 README.finn
```

Для специальных файлов устройств, именованных каналов и сокетов право x не определено, а права r и w стоит воспринимать как права ввода и вывода информации на устройство и как права передачи и приема информации через средство взаимодействия.

Дополнительные атрибуты

Помимо базовых прав доступа r, w и x, для решения отдельных задач разграничения доступа используют дополнительные атрибуты s, Set user/group ID (SUID Set User ID или SGID, Set Group ID) — атрибут неявного делегирования полномочий и t, sticky — «липучка», атрибут ограниченного удаления.

Типичной задачей, требующей неявного делегирования полномочий, является проблема невозможности изменения пользователями свойств своих учетных записей, которые хранятся в двух файлах-таблицах — passwd и shadow, доступных на запись (и чтение) только суперпользователю root.

Однако (листинг 3.40) команды passwd, chsh и chfn, будучи запущены обычным пользователем, прекрасно изменяют (!) пароль в таблице /etc/shadow и свойства пользовательской записи в таблице /etc/passwd за счет передачи полномочий владельца программы тому пользователю, который ее запускает.

Листинг 3.40. Дополнительный атрибут SUID

```
finn@ubuntu:~$ ls -la /etc/passwd /etc/shadow
-rw-r--r-- 1 root root 2867 ноя 17 11:16 /etc/passwd
-rw-r----- 1 root shadow 1617 ноя 17 11:17 /etc/shadow
finn@ubuntu:~$ passwd
Смена пароля для finn.
(текущий) пароль UNIX:
Ведите новый пароль UNIX:
Повторите ввод нового пароля UNIX:
passwd: пароль успешно обновлён
finn@ubuntu:~$ ls -la /etc/passwd /etc/shadow
-rw-r--r-- 1 root root 2867 ноя 17 11:16 /etc/passwd
-rw-r----- 1 root shadow 1617 ноя 18 01:46 * /etc/shadow
```

Листинг 3.40. Продолжение

```
finn@ubuntu:~$ chfn
```

Пароль:

Изменение информации о пользователе finn

Ведите новое значение или нажмите ENTER для выбора значения по умолчанию

Полное имя:

Номер комнаты []:

Рабочий телефон []: +7(812)703-02-02

Домашний телефон []:

```
finn@ubuntu:~$ ls -l /etc/passwd /etc/shadow
```

```
-rw-r--r-- 1 root root 2883 ноя 18 01:47 /etc/passwd
```

```
-rw-r----- 1 root shadow 1617 ноя 18 01:46 /etc/shadow
```

```
finn@ubuntu:~$ ls -la /usr/bin/passwd /usr/bin/chfn
```

```
-rwsr-xr-x 1 root root 84848 авг 29 16:00 /usr/bin/chfn
```

```
-rwsr-xr-x 1 root root 67992 авг 29 16:00 /usr/bin/passwd
```

За счет использования атрибута SUID получается, что пользователям, запускающим программы chfn, chsh и passwd, для их исполнения временно делегируются права владельца этих программ (суперпользователя root) так, как будто сам суперпользователь их запустил.

Листинг 3.41 Дополнительный атрибут SGID

```
Сеанс finn                                         tty1
finn@ubuntu:~$ w
00:03:53 up 12 days, 13:53, 7 users, load average: 0,53, 0,51, 0,91
USER     TTY      FROM          LOGIN@    IDLE      JCPU      PCPU  WHAT
jake     tty2          00:03    9.00s  0.52s  0.43s -bash
finn     tty1          00:03   17.00s  0.51s  0.45s -bash
finn@ubuntu:~$ ls -l /dev/tty1 /dev/tty2
crw----- 1 finn  tty 4, 1 окт. 20 00:03 /dev/tty1
crw----- 1 jake  tty 4, 2 окт. 20 00:03 /dev/tty2
finn@ubuntu:~$ write jake
write: jake has messages disabled
...
...
...
```

Листинг 3.41 Продолжение

Сеанс jake

tty2

```
jake@ubuntu:~$ mesg y
```

```
finn@ubuntu:~$ ls -l /dev/tty2
```

```
crw-rw---- 1 jake tty 4, 2 окт. 20 00:07 /dev/tty2
```

```
finn@ubuntu:~$ write jake
```

```
write: write: you have write permission turned off.
```

Hi, buddy, wazzup?

^D

Сеанс jake

tty2

```
jake@ubuntu:~$
```

```
:Message from finn@ubuntu on tty1 at 00:10 ...
```

Hey buddy, wazzup?

EOF

```
finn@ubuntu:~$ ls -ll /usr/bin/write
```

```
-rwxr-sr-x 1 root tty 14328 мая 3 2018 /usr/bin/write
```

Аналогично (см. листинг 3.41) при использовании атрибута SGID, при передаче сообщений от пользователя к пользователю командой write или wall, запускающему эти программы пользователю делегируются полномочия группы tty, имеющей доступ на запись к терминалам (специальным файлам устройств /dev/tty/Y), владельцы которых разрешили такой доступ.

Именно за счет механизма SUID/SGID различные команды позволяют обычным, непrivилегированным пользователям, выполнять сугубо суперпользовательские действия.

Так, например, su и sudo позволяют выполнять команды одним пользователям от лица других пользователей, mount, umount и fusermount — монтировать и размонтировать файловые системы, ping и traceroute — выполнять диагностику сетевого взаимодействия, at и crontab — сохранять в «системных» каталогах отложенные и периодические задания, и т. д.

Однако для каталогов атрибут SGID имеет совсем другой смысл.

По умолчанию владельцем файла становится тот пользователь (и его первичная группа), который запустил программу, создавшую файл.

Но для файлов, создаваемых в «общих» для какой-то группы пользователей, каталогах, логичнее было бы назначать группой-владельцем создаваемых файлов эту общую группу.

Листинг 3.41 Дополнительный атрибут SQID для каталога

```
bubblegum@ubuntu:~$ cd /srv/kingdom
bubblegum@ubuntu:/srv$ id
uid=1005(bubblegum) gid=1005(bubblegum) группы=1005(bubblegum),1007(candy)
bubblegum@ubuntu:/srv/kingdom$ ls -ld .
drwxr-xr-x 2 bubblegum bubblegum 4096 окт. 21 22:02 .
bubblegum@ubuntu:/srv/kingdom$ touch bananeguard1
bubblegum@ubuntu:/srv/kingdom$ ls -l .
итого 0
-rw-rw-r-- 1 bubblegum bubblegum 0 окт. 21 22:02 bananeguard1
bubblegum@ubuntu:/srv/kingdom$ chgrp candy .
bubblegum@ubuntu:/srv/kingdom$ chmod g+ws .
bubblegum@ubuntu:/srv/kingdom$ ls -ld .
drwxrwsr-x 2 bubblegum candy 4096 окт. 21 22:02 .
... ... ... ...
finn@ubuntu:/srv/kingdom$ id
uid=1001(finn) gid=1001(finn) группы=1001(finn),1007(candy)
finn@ubuntu:/srv/kingdom$ touch bananeguard2
finn@ubuntu:/srv/kingdom$ ls -l
итого 0
-rw-rw-r-- 1 bubblegum bubblegum 0 окт. 21 22:02 bananeguard1
-rw-rw-r-- 1 finn      candy      0 окт. 21 22:02 bananeguard2
```

В примере из листинга 3.42 за счет SGID-атрибута каталога владельцем всех файлов, помещаемых в этот каталог, автоматически назначается группа-владелец самого каталога, а создатель (владелец) файла может теперь назначать нужные права доступа для всех членов этой группы к своему файлу — либо неявно при помощи реверсивной маски доступа, либо явно при помощи команды chmod.

Атрибут-«липучка» t (sTicky) служит для ограничения действия базового разрешения w записи в каталоге.

Например, временный каталог /tmp предназначается для хранения временных файлов любых пользователей и поэтому доступен на запись всем пользователям.

Однако право записи в каталог дает возможность не только создавать в нем новые файлы, но и удалять любые существующие файлы (любых пользователей), что совсем не кажется логичным.

Именно атрибут t ограничивает возможность удалять чужие файлы, т. е. файлы, не принадлежащие пользователю, пытающемуся их удалить.

Листинг 3.43. Дополнительный атрибут sticky для каталога

```
finn@ubuntu:/srv/kingdom$ id  
uid=1001(finn) gid=1001(finn) группы=1001(finn),1007(candy)  
finn@ubuntu:/srv/kingdom$ ls -la  
итого 8  
drwxrwsr-x 2 bubblegum candy 4096 окт. 23 20:57 .  
drwxr-xr-x 3 root      root   4096 окт. 21 21:57 ..  
-rw-rw-r-- 1 bubblegum bubblegum 0 окт. 21 23:15 bananeguard1  
-rw-rw-r-- 1 finn      candy   0 окт. 21 23:24 bananeguard2  
...     ...     ...  
finn@ubuntu:/srv/kingdom$ rm bananeguard1  
rm: удалить защищенный от записи пустой обычный файл «bananeguard1»? у  
finn@ubuntu:/srv/kingdom$ ls -l  
итого 6  
-rw-rw-r-- 1 finn candy 0 окт. 21 23:24 bananeguard2  
...     ...     ...  
bubblegum@ubuntu:/srv/kingdom$ chmod +t .  
bubblegum@ubuntu:/srv/kingdom$ touch bananeguard1  
bubblegum@ubuntu:/srv/kingdom$ ls -la  
итого 8  
drwxrwsr-t 2 bubblegum candy 4096 окт. 23 21:19 .  
drwxr-xr-x 3 root      root   4096 окт. 21 21:57 ..  
-rw-rw-r-- 1 bubblegum candy   0 окт. 23 21:19 bananeguard1  
-rw-rw-r-- 1 finn      candy   0 окт. 23 21:19 bananeguard2  
...     ...     ...  
finn@ubuntu:/srv/kingdom$ rm bananeguard1  
rm: невозможно удалить «bananeguard1»: Операция не позволена ^
```

Списки контроля доступа POSIX

Режим доступа к файлу (access mode), определяющий базовые разрешения г, w и x только для трех субъектов доступа (владельца, группы-владельца и всех остальных), не является достаточно гибким и удобным инструментом разграничения доступа.

Списки контроля доступа (W:[ACL], access control lists), согласно стандарту POSIX. 1e, расширяют классический режим доступа к файлу дополнительными записями (рис. 3.4), определяющими права доступа для явно указанных пользователей и групп.

Для просмотра и модификации записей в списках доступа используются утилиты `getfacl` и `setfacl`, соответственно.

В примере из листинга 3.44 для всех «остальных» (не входящих в группу `candy`) пользователей отзываются все права на каталог `/srv/kingdom/stash`, но для отдельного пользователя `jake` (не являющегося членом группы `candy`) назначаются права чтения, модификации и прохода в него `rwx`.

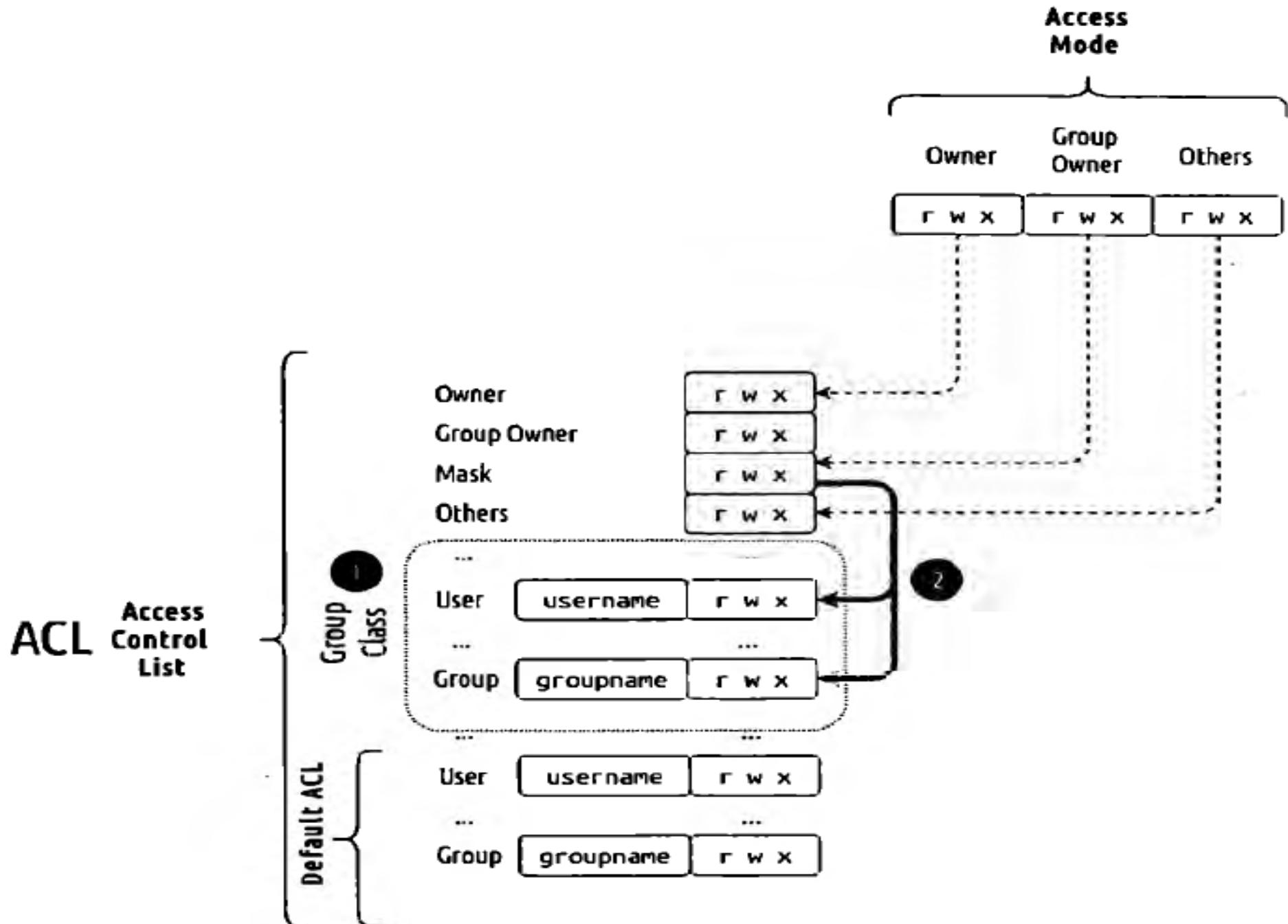


Рис. 3.4. Списки контроля доступа к файлам

Листинг 3.44. Списки контроля доступа

```
finn@ubuntu:/srv/kingdom$ ls -lad .
drwxrwsr-t 2 bubblegum candy 4096 окт. 23 21:19 .

finn@ubuntu:/srv/kingdom$ id
uid=1001(finn) gid=1001(finn) группы=1001(finn),1007(candy)
finn@ubuntu:/srv/kingdom$ mkdir stash
finn@ubuntu:/srv/kingdom$ chmod o= stash/
finn@ubuntu:/srv/kingdom$ ls -lad stash/
drwxrws--- 2 finn candy 4096 нояб. 4 13:05 stash/
finn@ubuntu:/srv/kingdom$ id jake
uid=1002(jake) gid=1002(jake) группы=1002(jake)
finn@ubuntu:/srv/kingdom$ setfacl -m u:jake:rwx stash/
finn@ubuntu:/srv/kingdom$ ls -lad stash/
drwxrws---+ 2 finn candy 4096 нояб. 4 13:05 stash/
finn@ubuntu:/srv/kingdom$ getfacl stash/
user::rwx
user:jake:rwx
group::rwx
mask::rwx
other::---
```

Групповая маска

С расширением множеств субъектов, для которых определены права доступа при помощи списков контроля доступа, возникает вопрос об их смысловой совместимости с режимом доступа, в котором определены всего три множества: пользователь-владелец, группа-владелец и все остальные.

Программы, работающие в соответствии с режимом доступа, считают, что если и существует некоторое количество «выделенных» субъектов со специально определенными правами, отличными от «всех остальных», то все эти субъекты входят в группу владельцев.

Списки контроля доступа позволяют «выделять» субъектов из числа любых пользователей и групп и определять их права произвольным образом.

Когда программа, работающая в соответствии с режимом доступа, назначает права группе-владельцу, она вправе считать, что все «выделенные» субъекты будут ограничены этими правами.

Именно поэтому в список контроля доступа добавлена групповая маска прав, определяющая ограничения «выделенных» субъектов (называемых групповым классом субъектов) в их индивидуальных правах.

В примере из листинга 3.45 каталогу, которому определены индивидуальные права rwx для пользователя jake в списке контроля доступа, изменяют права группы- владельцев классического режима доступа.

Получившийся режим доступа означает, что никому, кроме владельца файла, не разрешено записывать в этот файл, что противоречит индивидуальным правам списка доступа.

Противоречия устраняются маской списка доступа, ограничивающей эффективные права пользователя jake так, чтобы это соответствовало режиму доступа по смыслу.

Листинг 3.45. Групповая маска ACL

```
finn@ubuntu:/srv/kingdom$ ls -lad stash/
drwxrws---+ 2 finn candy 4096 нояб. 4 13:05 stash/
finn@ubuntu:/srv/kingdom$ getfacl stash/
...
user::rwx
user:jake:rwx
group::rwx
mask::rwx
other::---
finn@ubuntu:/srv/kingdom$ chmod g-w stash/
finn@ubuntu:/srv/kingdom$ ls -lad stash/
drwxr-s---+ 2 finn candy 4096 нояб. 4 13:05 stash/
finn@ubuntu:/srv/kingdom$ getfacl stash/
...
user::rwx
user:jake:rwx          #effective:r-x
group::rwx            #effective:r-x
mask::r-x
other::---
```

Права по умолчанию

При создании новых файлов в каталогах с индивидуальными правами пользователей в списках доступа зачастую складывается ситуация, когда пользователи (имеющие доступ в каталоги) не получают нужных прав доступа к создаваемым файлам в этих каталогах.

В большинстве случаев это противоречит здравому смыслу, т. к. все файлы некоторого каталога являются в определенном смысле «общими» для множества пользователей, которым разрешен доступ в сам каталог.

В примере из листинга 3.46 в каталоге `stash`, куда пользователю `jake` предоставлен индивидуальный доступ (см. листинг 3.45) при создании файла `README`, он в силу SCID для каталога (см. листинг 3.42) передается группе `candy`.

В группу `candy` пользователь `jake` не входит (именно поэтому ему назначены индивидуальные права в листинге 3.44), в результате чего файл ему никак не будет доступен.

Проблема решается назначением каталогу `stash` прав доступа «по умолчанию» (`default`), которые будут унаследованы файлами, создающимися в этом каталоге.

Листинг 3.46 Права по умолчанию

```
finn@ubuntu:/srv/kingdom$ cd stash/
finn@ubuntu:/srv/kingdom$ umask 0007
finn@ubuntu:/srv/kingdom/stash$ touch README
finn@ubuntu:/srv/kingdom/stash$ ls -la README
-rw-rw---- 1 finn      candy 0 нояб. 4 14:16 README
finn@ubuntu:/srv/kingdom/stash$ id jake
uid=1002(jake) gid=1002(jake) группы=1002(jake)
finn@ubuntu:/srv/kingdom/stash$ setfacl -m d:u:jake:rwx .
finn@ubuntu:/srv/kingdom/stash$ getfacl .
# file: .
...
default:user::rwx
```

Листинг 3.46 Продолжение

```
default:user:jake:rw-
default:group::rwx
default:mask::rwx
default:other::---
finn@ubuntu:/srv/kingdom/stash$ touch README.jake
finn@ubuntu:/srv/kingdom/stash$ ls -l README.jake
-rw-rw----+ 1 finn      candy    0 нояб. 4 14:17 README.jake
finn@ubuntu:/srv/kingdom/stash$ getfacl README.jake
# file: README.jake
...
user:jake:rw-
...
...
```

Мандатное (принудительное) разграничение доступа

В Linux дискреционные механизмы разграничения доступа (DAC, discretionary access control) являются основными и всегда активны.

Их использование предполагает, что владельцы объектов правильно распоряжаются правами доступа к находящимся в их владении объектам.

Например, пользовательские закрытые ключи, используемые службой W:[SSH] (см. разд. 6.4.1) в каталоге `~/.ssh` или ключи W: [СпиРС] в каталоге `~/.gnupg` и прочие секретные данные (подобные ключи доступа в банковские информационные системы), должны быть недоступны никому, кроме их владельца.

Запускаемые пользователем программы выполняются от лица запустившего их пользователя и имеют доступ к файлам согласно установленным режимам или спискам доступа.

В примере из листинга 3.47 клиент `ssh`, браузер `firefox` и коммуникатор `skype` имеют абсолютно равные возможности по чтению и модификации (!) пользовательского закрытого ключа `~/.ssh/id_rsa`, тогда как настоящим «владельцем» ключей является только `sshfl`).

Листинг 3.47. Необходимость MAC

```
finn@ubuntu:~$ ls -l .ssh
```

итого 8

```
-rw----- 1 finn  finn 1675 нояб. 4 16:06 id_rsa  
-rw-r--r-- 1 finn      finn   393 нояб. 4 16:06 id_rsa.pub
```

```
finn@ubuntu:~$ ps fux
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
finn	20650	0.0	0.1	13488	8576	pts/0	S	16:08	0:00	-bash
finn	20943	0.0	0.0	6668	2284	pts/0	S+	16:19	0:00	\ ssh jake@grex.org
						
finn	21094	13.1	1.5	592676	127820	pts/2	Rl+	16:27	0:09	/usr/lib/firefox/firefox
						
finn	21790	29.7	2.1	575000	172084	?	Sl	16:38	0:09	skype

Абсолютно естественно предполагать, что программы firefox и skype не имеют никаких намерений доступа к пользовательским ключам SSH.

Можно даже доверять программе firefox, штатно установленной из доверенного источника (дистрибутива), где она была изготовлена из открытых исходных текстов, подлежащих верификации.

Однако нет никаких оснований доверять закрытому skype, поставляемому в бинарном виде.

Более того, предоставлять доступ программам firefox и skype к SSH-ключам пользователя нет никакой необходимости, во-первых, просто потому, что это выходит за рамки набора минимально необходимых условий их целевого функционирования.

Во-вторых, практически в любой программе есть ошибки, используя которые злоумышленник может осуществлять непреднамеренные действия в свою пользу.

Таким воздействиям особенно подвержены программы, использующие сетевой обмен с недоверенной внешней средой — клиенты и серверы сетевых служб операционной системы.

Тем временем дискреционный подход и механизмы служат для разграничения доступа разных пользователей к файлам, но никак не предназначены для разграничения доступа программ одного и того же пользователя к разным файлам этого пользователя.

Для разграничения доступа субъектов — программ к объектам — файлам дерева каталогов используют так называемый мандатный (от англ, mandatory — обязательный или принудительный) подход (MAC, mandatory access control), предполагающий следование обязательным правилам доступа к файлам, назначаемым администраторами системы.

Правила доступа строятся на основе знания о внутреннем устройстве программ и представляют собой описание набора минимально необходимых условий их целевого функционирования.

Таким образом, в мандатных правилах, ограничивающих доступ к SSH-ключам пользователя, только programme ssh должен быть разрешен доступ для непосредственного выполнения своих прямых функций, а программам firefox и skype в доступе к SSH-ключам должно быть отказано.

Модуль принудительного разграничения доступа AppArmor

В Linux мандатные механизмы разграничения являются дополнительными и активируются по желанию пользователя или дистрибутора.

Так, например, в Ubuntu Linux по умолчанию устанавливается и активируется модуль принудительного разграничения доступа AppArmor.

Модуль W: [AppArmor] имеет некоторое количество готовых к употреблению наборов мандатных правил (называемых профилями) для ограничения (confine) доступа субъектов — программ к объектам операционной системы — файлам, сетевым протоколам, портам TCP/UDP и пр.

Правила AppArmor идентифицируют программы и файлы на основе их полных путевых имен.

При этом каждый профиль описывает ограничения для одной определенной запускаемой программы (а также для других, запускаемых этой программой, т. е. «подчиненных» программ), а программы, для которых профили не определены, никак не ограничиваются (unconfined).

В примере из листинга 3.48 проиллюстрировано использование команды `aa-status`, показывающей статус модуля принудительного контроля доступа AppArmor.

В распоряжении модуля имеются профиль программы `skype` в режиме принуждения (`enforce`) и профиль программы `firefox` в режиме оповещения (`complain`).

Более того, процесс `skype` с идентификатором 30173 принуждается (`enforce`) модулем контроля к выполнению правил профиля, а процесс `firefox` с идентификатором 10335 только отслеживается (`complain`) на предмет нарушений правил.

Листинг 3.48. Модуль мандатного разграничения доступа AppArmor

```
finn@ubuntu:~$ sudo aa-status
apparmor module is loaded.
50 profiles are loaded.
26 profiles are in enforce mode.
    /usr/bin skype
        ...
        ...
24 profiles are in complain mode.
    ...
    ...
    /usr/lib/firefox/firefox{,*[^s][^h]}
        ...
        ...
18 processes have profiles defined.
3 processes are in enforce mode.
    ...
    ...
    /usr/bin skype (30173)
6 processes are in complain mode.
    ...
    ...
    /usr/lib/firefox/firefox{,*[^s][^h]} (10335)
        ...
        ...
0 processes are unconfined but have a profile defined.
```

Нарушения мандатных правил процессами, находящимися в режиме оповещения (`complain`), приводят только к журнализации сообщений аудита об обнаруженных нарушениях.

Попытки нарушения мандатных правил процессами, находящимися в режиме принуждения (`enforce`), пресекаются модулем контроля доступа в виде отказа в доступе к тому или иному запрашиваемому объекту.

Выполненный (листинг 3.49) при помощи команды `apparmor_parser` анализ полного (-р) набора правил профиля программы `/usr/bin/firefox` показывает, что обращения к любым файлам каталога `.ssh` явно запрещены указанием `deny` и подлежат обязательному аудиту согласно указанию `audit`.

При помощи команды `aa-enforce` профиль переводится в режим `enforce`, в результате чего доступ `firefox` к ключам пользователя оказывается запрещенным.

Листинг 3.49 Мандатные правила профиля Firefox

```
finn@ubuntu:~$ apparmor_parser -p /etc/apparmor.d/usr.bin.firefox
```

```
... ... ... ... ...  
audit deny /* @{{HOME}}/.ssh/** mgnkl,
```

```
... ... ...
```

```
finn@ubuntu:~$ cat ~/.ssh/id_rsa
```

```
-----BEGIN RSA PRIVATE KEY-----
```

```
... ... ...
```

```
-----END RSA PRIVATE KEY-----
```

```
finn@ubuntu:~$ firefox ~/.ssh/id_rsa
```

Окно firefox

```
file:///home/finn/.ssh/id_rsa
```

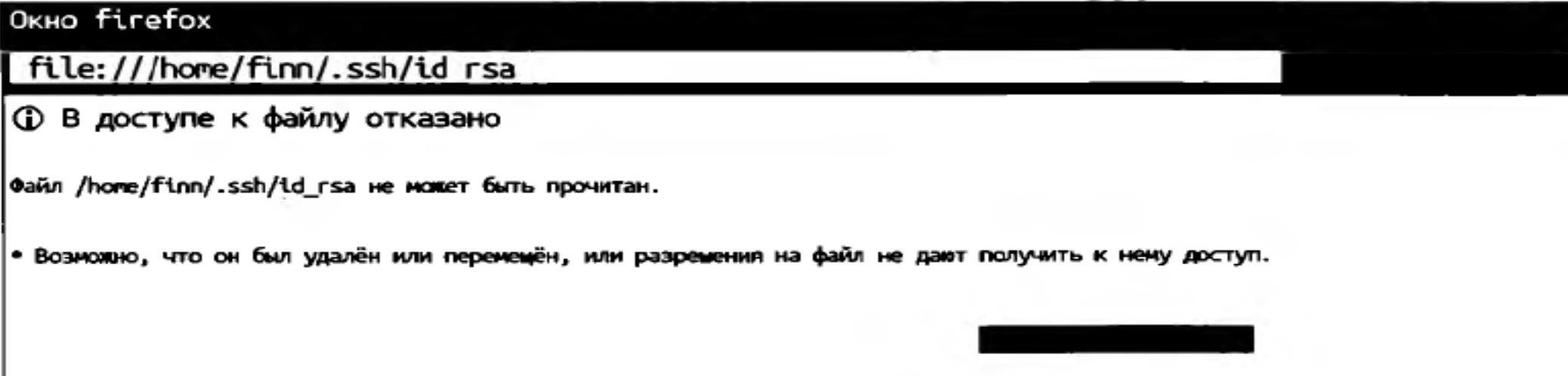
```
-----BEGIN RSA PRIVATE KEY-----
```

```
... ... ...
```

```
-----END RSA PRIVATE KEY-----
```

Листинг 3.49 Продолжение

```
finn@ubuntu:~$ sudo aa-enforce firefox
Profile for /usr/lib/firefox/firefox.sh not found, skipping
finn@ubuntu:~$ pgrep firefox
16392
finn@ubuntu:~$ ps up 16392
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
finn  16392  2.4  5.9 2974256 238292 pts/2  Sl   23:33  0:03 /usr/lib/firefox/firefox
finn@ubuntu:~$ sudo aa-enforce /usr/lib/firefox/firefox
Назначение /usr/lib/firefox/firefox принудительного режима.
Warning: profile /usr/lib/firefox/firefox{,[^s][^h]} represents multiple programs
finn@ubuntu:~$ pkill firefox
...
...
finn@ubuntu:~$ firefox ~/.ssh/id_rsa
```



Стоит отметить, что команды aa-enforce и aa-status выполняются от лица суперпользователя, единолично управляющего модулем принудительного контроля доступа AppArmor, но детальное рассмотрение синтаксиса правил и процедур управления модулем относится к задачам администрирования операционной системы

Дополнительные свойства файлов

Расширенные атрибуты файлов

Как было показано выше (см. листинг 3.4), базовые права доступа, дополнительные атрибуты SUID/Sgid, владельцы, счетчик имен и. другие основные свойства файлов хранятся в их метаданных.

Кроме этого, файлам могут быть назначены списки контроля доступа (см. листинг 3.44), которые являются их дополнительными свойствами и хранятся за пределами метаданных, при помощи расширенных атрибутов attr.

Каждый расширенный атрибут имеет имя вида namespace.attrname, при этом пространства имен namespace определяют назначение атрибута.

Пространство имен system используется системными (ядерными) компонентами, например, для списков контроля доступа POSIX ACL.

Пространство имен security используется системными компонентами безопасности, в частности для хранения привилегий исполняемых программ (capabilities).

Пространства имен trusted и user предназначены для атрибутов внеядерных компонент — программ, выполняющихся привилегированным и обычными пользователями, соответственно.

Для просмотра и назначения внеядерных (пользовательских) расширенных атрибутов используются утилиты `getfattr` и `setfattr`. Читать пользовательские расширенные атрибуты файла разрешено тем же субъектам, которым разрешено чтение данных этого файла.

Аналогично, устанавливать (изменять) и удалять пользовательские расширенные атрибуты файла могут субъекты, допущенные к записи данных этого файла.

Ядерные (системные) атрибуты обычно управляются специально предназначеными командами: например, `getfacl` и `setFacl` предназначены для списков контроля доступа, команды `getcap` и `setcap` — для привилегий исполняемых программ, команды `chcon` и `ls-Z` — для мандатных меток.

Системные атрибуты, как правило, всегда доступны для чтения, но их установка и изменение требуют определенных привилегий процесса.

Листинг 3.56. Расширенные системные атрибуты файлов

```
finn@ubuntu:~$ getcap /usr/bin/gnome-keyring-daemon
/usr/bin/gnome-keyring-daemon = cap_ipc_lock+ep ①

finn@ubuntu:~$ getfattr -d -m - /usr/bin/gnome-keyring-daemon
getfattr: Удаление начальных '//' из абсолютных путей
# file: /usr/bin/gnome-keyring-daemon
security.capability=0sAQAAAgaAAAAAAAAAAAAAAA=
```

```
finn@ubuntu:~$ cd /srv/kingdom/stash
finn@ubuntu:/srv/kingdom/stash$ ls -l README.jake
-rw-rw----+ 1 finn      candy    0 нояб.  4 14:17 README.jake
finn@ubuntu:/srv/kingdom/stash$ getfacl README.jake
# file: README.jake
# owner: finn
# group: candy
user::rw-
user:jake:rw- ②
group::rwx          #effective:rw-
mask::rw-
other::---
```

Листинг 3.56. Продолжение

```
finn@ubuntu:/srv/kingdom/stash$ getfattr -d -n - README.jake  
# file: README.jake  
system.posix_acl_access=0sAgAAAAEABgD////AgACAOoDAAAEEAACA////xAABgD////IAAAAP///8=
```

```
finn@ubuntu:/srv/kingdom/stash$ setfattr -n user.color -v orange README.jake  
finn@ubuntu:/srv/kingdom/stash$ setfattr -n user.flavour -v vanilla README.jake  
finn@ubuntu:/srv/kingdom/stash$ getfattr -d README.jake  
# file: README.jake  
user.color="orange"  
user.flavour="vanilla"
```

В примере из листинга 3.56 показано, что привилегии исполняемых программ на самом деле сохранены в атрибуте security.capability, списки контроля доступа — в атрибуте system.posix_acl_access, а в атрибутах пространства имен user можно разместить любые значения.

Наиболее известным применением пользовательских атрибутов (листинг 3.57) являются атрибуты user.xdg.origin.url и

user.xdgreferrer.url, используемые браузером chromium-browser для сохранения URL файлов, которые были загружены из Интернета.

Листинг 3.57 Расширенные пользовательские атрибуты файлов

```
flnn@ubuntu:~/Downloads$ getfattr -d TLCL-13.07.pdf
```

```
# file: TLCL-13.07.pdf
```

```
user.xdg.origin.url="http://freefr.dl.sourceforge.net/project/linuxcommand/TLCL/13.07/TLCL-13.07.pdf"
```

```
user.xdg.referrer.url="http://sourceforge.net/projects/linuxcommand/files/TLCL/13.07/TLCL-13.07.pdf/download"
```

Флаги файлов

Кроме «общих» расширенных атрибутов, которые используются разными компонентами операционной системы, каждая файловая система зачастую имеет собственные атрибуты файлов, управляющие ее поведением и функциями при доступе к файлам.

Так, файловые системы ext2/ext3/ext4 управляются специальными атрибутами-флагами, например a (append only), i (immutable), s (secure deletion), S (synchronous updates) и пр.

Флаг *s* заставляет файловую систему при удалении файла не только высвобождать принадлежащие ему блоки, но и обнулять их, а флаг *S* заставляет операции записи в файл выполнятся синхронно (немедленно), минуя отложенную запись с использованием дискового кэша.

Флаг *i* делает файл «неприкасаемым» (листинг 3.58) — его нельзя ни изменить в, ни удалить никому, даже суперпользователю *root*.

Флаг *a* делает файл «накопительным», т. е. никому не дает изменять имеющуюся в файле информацию или удалять файл, а позволяет только добавлять данные в его конец.

Устанавливать флаги файлов разрешено их владельцам, а установка отдельных флагов, например *a* или *I*, требует определенных привилегий процесса (см. разд. 4.5.2).

Для просмотра флагов файлов предназначена утилита *lsattr*, а для их изменения — утилита *chattr*, что иллюстрирует листинг 3.58 на примере флага *I*.

Листинг 3.58. Флаги файлов

```
finn@ubuntu:/srv/kingdom/stash$ lsattr  
-----e- ./README.jake  
-----e- ./README  
finn@ubuntu:/srv/kingdom/stash$ chattr +i README.jake  
chattr: Операция не позволяет while setting flags on README.jake  
finn@ubuntu:/srv/kingdom/stash$ sudo chattr +i README.jake  
finn@ubuntu:/srv/kingdom/stash$ lsattr README.jake  
----i-----e- README.jake  
finn@ubuntu:/srv/kingdom/stash$ date >1 README.jake  
-bash: README.jake: Операция не позволяет  
finn@ubuntu:/srv/kingdom/stash$ rm README.jake  
rm: невозможно удалить «README.jake»: Операция не позволяет  
finn@ubuntu:/srv/kingdom/stash$ sudo rm README.jake  
rm: невозможно удалить «README.jake»: Операция не позволяет
```

В заключение

Всестороннее рассмотрение разнообразных файлов, их свойств, атрибутов и контекстов использования неизбежно должно приводить к выводу, что файл является универсальной сущностью, позволяющей организовать однородный доступ к информации, вне зависимости от свойств ее источника.

Специальные файлы устройств, именованные каналы и сокеты имеют файловую природу и могут обрабатываться совершенно «обычными» программами за счет идентичности их файлового программного интерфейса, наравне с файлами «обычных» (дисковых), сетевых, псевдофайловых и внеядерных файловых систем.

Подсистема управления процессами, о которой пойдет речь в следующем материале, тоже не обходится без файлов и использует механизм их отображения в память для организации виртуальной памяти и средств межпроцессного взаимодействия, таких как разделяемая память, очереди сообщений¹ и семафоры.

Даже сетевые сокеты, рассмотрение которых отложено далее, тоже на поверку оказываются файлами.

Таким образом, понимание файла как основополагающей компоненты операционной системы дает ключ к пониманию многих других ее частей, а навыки мониторинга файлов или трассировки файлового интерфейса позволяют заглянуть в корень практически всех ее механизмов.

Управление процессами и памятью

Лекция 5 +

Процессы операционной системы в большинстве случаев отождествляются с выполняющимися программами, что не совсем верно, точнее — совсем не верно.

В современных операционных системах, включая Linux, между программой и процессом есть очевидная взаимосвязь, но далеко не такая непосредственная, как кажется на первый взгляд.

Программы и библиотеки

Программа представляет собой алгоритм, записанный на определенном языке, понятном исполнителю программы.

Различают машинный язык, понятный центральному процессору, и языки более высоких уровней (алгоритмические), понятные составителю программы — программисту.

Программы, составленные на языке высокого уровня, в любом случае перед исполнением должны быть транслированы (переведены) на язык исполнителя, что реализуется при помощи специальных средств — трансляторов.

Различают два вида трансляторов программ — компиляторы и интерпретаторы. Компилятор транслирует в машинный код сразу целиком всю программу и не участвует в ее исполнении.

Интерпретатор, наоборот, пошагово транслирует отдельные инструкции программы и немедленно выполняет их.

Например, командный интерпретатор при интерактивном режиме пошагово выполняет команды, вводимые пользователем, а в пакетном режиме так же пошагово выполняет команды, записанные в файле сценария.

Алгоритм, в свою очередь, есть некоторый набор инструкций, выполнение которых приводит к решению конкретной задачи.

В большинстве случаев инструкции алгоритма имеют причинно-следственные зависимости и выполняются исполнителем последовательно.

Однако если выделить «независимые» поднаборы инструкций (независимые ветви), то их можно выполнять несколькими исполнителями одновременно — параллельно.

Поэтому различают последовательные и параллельные алгоритмы и соответствующие им последовательные и параллельные программы.

Некоторые программы реализуют алгоритмы общего назначения, например алгоритмы сжатия или шифрования информации, алгоритмы сетевых протоколов и т. д.

Такие программы, востребованные не только конечными пользователями, сколько другими программами, называют библиотеками.

Согласно hier, откомпилированные до машинного языка программы размещаются в каталогах

/bin,

/sbin,

/usr/bin,

/usr/sbin,

/usr/local/bin,

/usr/local/sbin,

а библиотеки — в каталогах

/lib, /usr/lib, /usr/local/lib.

Программы имеют специальный бинарный «запускаемый» формат W: [ELF] executable и зависят от библиотек, что проиллюстрировано в листинге 4.1 при помощи команды ldd (loader dependencies).

Каждая зависимость отображается именем библиотеки (SONAME, shared object name), найденным в системе файлом библиотеки и адресом в памяти процесса (32- или 48-битным, в зависимости от платформы), куда библиотека будет загружена.

Листинг 4.1. Программы и библиотеки

```
fitz@ubuntu:~$ which ls
/usr/bin/ls
fitz@ubuntu:~$ file /usr/bin/ls
/usr/bin/ls: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked,
interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=2f15ad836be3339dec0e2e6a3c637e08e48aacbd, for GNU/Linux 3.2.0, stripped
fitz@ubuntu:~$ ldd /usr/bin/ls
    linux-vdso.so.1 (0x00007ffcb529d000)
    libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1 (0x00007fb02f58d000)
❶ libc.so.6 => ❷ /lib/x86_64-linux-gnu/libc.so.6 (0x00007fb02f39c000) ❸
    libpcre2-8.so.0 => /lib/x86_64-linux-gnu/libpcre2-8.so.0 (0x00007fb02f317000)
    libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007fb02f311000)
    /lib64/ld-linux-x86-64.so.2 (0x00007fb02f5f1000)
    libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007fb02f2ee000)

fitz@ubuntu:~$ file /lib/x86_64-linux-gnu/libc.so.6
/lib/x86_64-linux-gnu/libc.so.6: symbolic link to libc-2.30.so
```

Нужно заметить, что файла библиотеки `linux-vdso.so.1` (реализующей интерфейс системных вызовов к ядру) не существует, т. к. она является виртуальной (VDSO, *virtual dynamic shared object*), т. е. предоставляется и отображается в память процесса самим ядром, «как будто» является настоящей библиотекой.

Кроме того, библиотека `ld-linux-x86-64.so.2` указана абсолютным путевым именем, поэтому поиск ее файла не производится.

Для большинства библиотек зависимость устанавливается при помощи SONAME вида `libNAME.so.X`, где `lib` — стандартный префикс (*library*, библиотека), `.so` — суффикс (*shared object*, разделяемый объект), `NAME` — имя «собственное», а `.X` — номер версии ее интерфейса (листинг 4.2).

По имени SONAME в определенных (конфигурацией компоновщика — см. `ld.so` и `ldconfig`) каталогах производится поиск одноименного файла библиотеки, который на самом деле оказывается символической ссылкой на «настоящий» файл библиотеки.

Например, для б-й версии интерфейса динамической библиотеки языка с (`ltbc.so.6`) настоящий файл библиотеки называется `libc-2.30.so`, что указывает на версию самой библиотеки как 2.30.

Листинг 4.2. Версии библиотек

```
fttz0ubuntu:~$ file /lib/x86_64-linux-gnu/libpcre2-8.so.0  
/lib/x86_64-linux-gnu/libpcre2-8.so.0: symbolic link to libpcre2-8.so.0.7.1
```

Аналогично, в листинге 4.2 показано, что для 0-й версии интерфейса динамической библиотеки регулярных perl-выражений pcre2 (libpcre2-8.so.0) настоящий файл библиотеки называется libpcre2-8.so.0.7.1, а это указывает на версию самой библиотеки как 0.7.1.

Такой подход позволяет заменять (исправлять ошибки, улучшать - неэффективные алгоритмы и пр.) библиотеки (при условии неизменности их интерфейсов) отдельно от программ, зависящих от них.

При обновлении библиотеки libc-2.30.so, например, до libc-2.32.so достаточно установить символьическую SONAME-ссылку libc.so.6 на libc-2.32.so, в результате чего ее начнут использовать все программы с зависимостями от libc.so.6.

Более того, в системе может быть одновременно установлено любое количество версий одной и той же библиотеки, реализующих одинаковые или разные версии интерфейсов, выбор которых будет указан соответствующими SONAME-ссылками.

Листинг 4.3. библиотеки—это незапускаемые программы

```
fitz@ubuntu:~$ file /lib/x86_64-linux-gnu/libc-2.30.so
```

```
/lib/x86_64-linux-gnu/libc-2.30.so: ELF 64-bit LSB shared object, x86-64, version 1  
(GNU/Linux), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,  
BuildID[sha1]=2155f455ad56bd871c8225bcc85ee25c1c197c4, for GNU/Linux 3.2.0, stripped
```

```
fitz@ubuntu:~$ file /lib/x86_64-linux-gnu/libpcre2-8.so.0.7.1
```

```
/lib/x86_64-linux-gnu/libpcre2-8.so.0.7.1: ELF 64-bit LSB shared object, x86-64, version 1  
(SYSV), dynamically linked, BuildID[sha1]=815e1acbcc22015f05d62c17fe982c1b573125b1, stripped
```

```
fitz@ubuntu:~$ ldd /lib/x86_64-linux-gnu/libpcre2-8.so.0.7.1
```

```
linux-vdso.so.1 (0x00007ffe22093000)
```

```
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f8ec2bdd000)
```

```
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f8ec29ec000)
```

```
/lib64/ld-linux-x86-64.so.2 (0x00007f8ec2c99000)
```

Библиотеки имеют тот же бинарный формат W:[ELF], что и «запускаемые» программы, но не «запускаемый» executable, а «совместно используемый» shared object.

Библиотеки, являясь пусты и незапускаемыми, но программами, естественным образом тоже зависят от других библиотек, что показано в листинге 4.3.

Практически «запускаемость» ELF-файлов (листинг 4.4) зависит не от их типа, а от прав доступа и осмысленности точки входа — адреса первой инструкции, которой передается управление при попытке запуска.

Например, библиотеку libc-2.30.so можно запустить, в результате чего будет выведена статусная информация

Листинг 4.4. Запускаемые библиотеки

```
fitz@ubuntu:~$ ls -l /lib/x86_64-linux-gnu/libc-2.30.so
-rwxr-xr-x 1 root root 2025032 сен 16 17:56 /lib/x86_64-linux-gnu/libc-2.30.so
fitz@ubuntu:~$ /lib/i386-linux-gnu/libc-2.15.so
GNU C Library (Ubuntu GLIBC 2.30-0ubuntu2) stable release version 2.30.
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.
Compiled by GNU CC version 9.2.1 20190909.
libc ABIs: UNIQUE IFUNC ABSOLUTE
For bug reporting instructions, please see:
<https://bugs.launchpad.net/ubuntu/+source/glibc/+bugs>.
```

Ядро Linux

Не стоит забывать, что самой главной программой операционной системы является ее ядро, которое в Linux состоит из статического стартового модуля (листинг 4.5) в формате ELF executable и динамически пристыковываемых программных модулей формата ELF relocatable (листинг 4.6).

Для выполнения процедуры начальной загрузки стартовый модуль упакован в «самораспаковывающийся» gzip-архив формата bzImage (big zipped image), который состоит из программы распаковки и собственно запакованного стартового модуля.

В листинге 4.5 проиллюстрирован процесс извлечения стартового модуля из архива /boot/vmlinuz-3.13.0-49-generic формата bzImage , который предварительно копируется в /tmp/vmlinuz.

Для извлечения используется сценарий extract-vmlinux из пакета заголовочных файлов ядра.

Распакованный стартовый модуль /tmp/vmlinuz ожидаемо оказывается статически скомпонованной (т. е. не использующей библиотеки ELF shared object) исполняемой ELF-программой.

Листинг 4.5. Ядро операционной системы

```
fitz@ubuntu:~$ uname -r
5.3.0-23-generic

fitz@ubuntu:~$ file /boot/vmlinuz-5.3.0-23-generic
/boot/vmlinuz-5.3.0-23-generic: regular file, no read permission

fitz@ubuntu:~$ ls -l /boot/vmlinuz-5.3.0-23-generic
-rw----- 1 root root 11399928 ноя 12 11:51 /boot/vmlinuz-5.3.0-23-generic

fitz@ubuntu:~$ sudo file /boot/vmlinuz-5.3.0-23-generic
/boot/vmlinuz-5.3.0-23-generic: Linux kernel x86 boot executable bzImage, version 5.3.0-23-
generic (buildd@lgw01-amd64-002) #25-Ubuntu SMP Tue Nov 12 09:22:33 UTC 2019, R0-rootFS,
swap_dev 0xA, Normal VGA

fitz@ubuntu:~$ sudo cat /boot/vmlinuz-5.3.0-23-generic > /tmp/vmlinuz

fitz@ubuntu:~$ /usr/src/linux-headers-5.3.0-23/scripts/extract-vmlinux /tmp/vmlinuz >
/tmp/vmlinux

fitz@ubuntu:~$ file /tmp/vmlinuz
/tmp/vmlinuz: ELF 64-bit LSB executable ③, x86-64, version 1 (SYSV), statically linked,
BuildID[sha1]=b23ff3f6790319ec53B278e3269af619ba2ca642, stripped
```

Динамические модули загружаются в пространство ядра и пристыковываются к стартовому модулю позднее, уже при работе операционной системы при помощи системных утилит `insmod` или `modprobe`.

Для отстыковки и выгрузки ненужных модулей предназначена системная утилита `rmmod`, для просмотра списка (см. листинг 4.6) загруженных модулей — `lsmod`, а для идентификации свойств и параметров модулей — утилита `modinfo`.

Загрузка и выгрузка модулей реализуется специальными системными вызовами `init_module` и `delete_module`, доступ к списку загруженных модулей — при помощи файла `/rgos/modules` псевдофайловой системы `rgos`, а идентификация свойств и параметров модулей — чтением специальных секций ELF-файлов модулей.

Листинг 4.6. Модули ядра

```
fitz@ubuntu:~$ lsmod
```

Module	Size	Used by
i915	1949696	4
btusb	57344	0
uvcvideo	98304	0
e1000e	258048	0

```
fitz@ubuntu:~$ modinfo i915
```

```
filename:      /lib/modules/5.3.0-23-generic/kernel/drivers/gpu/drm/i915/i915.ko
license:       GPL and additional rights
description:   Intel Graphics
```

```
fitz@ubuntu:~$ file /lib/modules/5.3.0-23-generic/kernel/drivers/gpu/drm/i915/i915.ko
/lib/modules/5.3.0-23-generic/kernel/drivers/gpu/drm/i915/i915.ko: ELF 64-bit LSB
relocatable, x86-64, version 1 (SYSV),
BuildID[sha1]=49e59590c1a718074b76b6541702f6f794ea7eae, not stripped
```

Динамические модули ядра зачастую являются драйверами устройств, что проиллюстрировано в листинге 4.7 при помощи утилит `lspci` и `lsusb`, которые сканируют посредством псевдофайловой системы `sysfs` списки обнаруженных ядром на шинах PCI и USB устройств и обслуживающих их драйверов.

Листинг 4.7. Драйверы устройств

```
fitz@ubuntu:~$ lspci -k
...
00:02.0 VGA compatible controller: Intel Corporation 2nd Generation Core Process
or Family Integrated Graphics Controller (rev 09)
    Subsystem: Dell 2nd Generation Core Processor Family Integrated Graphics
Controller
    Kernel driver in use: i915
    Kernel modules: i915
...
00:19.0 Ethernet controller: Intel Corporation 82579LM Gigabit Network Connection
(Lewisville) (rev 04)
    Subsystem: Dell 82579LM Gigabit Network Connection (Lewisville)
    Kernel driver in use: e1000e
    Kernel modules: e1000e

fitz@ubuntu:~$ lsusb -t
...
/: Bus 01.Port 1: Dev 1, Class=root_hub, Driver=ehci-pci/3p, 480M
|__ Port 1: Dev 2, If 0, Class=Hub, Driver=hub/6p, 480M
    |__ Port 4: Dev 3, If 2, Class=Vendor Specific Class, Driver=, 12M
    |__ Port 4: Dev 3, If 0, Class=Wireless, Driver=btusb, 12M
    |__ Port 4: Dev 3, If 3, Class=Application Specific Interface, Driver=, 12M
    |__ Port 4: Dev 3, If 1, Class=Wireless, Driver=btusb, 12M
    |__ Port 5: Dev 4, If 0, Class=Video, Driver=uvcvideo, 480M
    |__ Port 5: Dev 4, If 1, Class=Video, Driver=uvcvideo, 480M
```

Процессы и нити

Сущность процесса неразрывно связана с мультипрограммированием и многозадачностью операционной системы.

Например, в однозадачных операционных системах программы существуют, а процессы — нет.

В однозадачных операционных системах единовременно одна последовательная программа выполняется одним исполнителем (центральным процессором), имея возможность безраздельно использовать все доступные ресурсы (память, устройства ввода-вывода и пр.).

В любой программе можно выделить перемежающиеся блоки инструкций, использующих или центральный процессор (ЦП), или устройства ввода-вывода (УВВ).

При этом центральный процессор вынужден простоять при выполнении программой операций ввода-вывода, например, при ожидании окончания записи (или чтения) блока данных на внешний носитель, или при ожидании окончания передачи или приема сетевого кадра, или при ожидании событий с устройств человеко- машинного взаимодействия.

С другой стороны, устройства ввода-вывода тоже вынуждены простоять при выполнении программой вычислительных операций, например ожидая результата, подлежащего выводу, или ожидая возникновения > программы потребности в новых исходных данных.

Используя такую модель поведения программ, можно провести анализ потребления ими ресурсов при выполнении.

Например, компрессоры gzip, bzip и xz считывают очередной блок данных исходного файла, относительно долго упаковывают его и записывают в результирующий файл, а затем повторяют процедуру до исчерпания блоков исходного файла.

Количество времени, потраченного на вычислительные операции упаковки, будет много больше количества времени, потраченного на чтение исходных данных и запись результатов, поэтому нагрузка на ЦП будет высокой, а на УВВ — нет.

Такой же анализ можно привести и для дубликатора dd, копировщика rsync или архиватора tar, которые, наоборот, почти не выполняют никаких вычислений, а сосредоточены на вводе-выводе больших объемов данных, поэтому при их использовании нагрузка на ЦП будет довольно низкой, а на УВВ — высокой.

Для командного интерпретатора bash, текстовых редакторов nano и vim и других интерактивных программ, взаимодействующих с пользователем, характерны длительные ожидания ввода небольших команд, простая и недолгая их обработка и вывод короткого результата.

В результате коэффициент полезного использования и ЦП, и УВВ будет приближен к нулю.

Подобный анализ и желание увеличить коэффициенты полезного использования ресурсов привели к созданию многозадачных операционных систем, основывающихся на простой идее псевдоодновременного выполнения нескольких последовательных программ одним исполнителем.

Для этого вместоостоя в ожидании окончания операции ввода-вывода, начатой некоторой программой, центральный процессор переключается на выполнение другой программы, тем самым увеличивая интегральный коэффициент его полезного использования.

С появлением мультипрограммной смеси каждая из ее программ больше не может безраздельно использовать все доступные ресурсы (например, всю память — она одновременно нужна всем программам смеси), в связи с чем операционная система берет на себя задачи диспетчеризации (распределения) ресурсов между ними.

В Linux, как и во многих других операционных системах, программы изолируются друг от друга в специальных «виртуальных» средах, обеспечивающих их процесс выполнения.

Каждая такая среда называется процессом и получает долю доступных ресурсов — выделенный участок памяти, выделенные промежутки процессорного времени.

Процесс эмулирует для программы «однозадачный» режим выполнения, словно программа выполняется в одиночку, и «безраздельное» использование ресурсов процесса, как будто это все доступные ресурсы.

Параллельные программы, как указывалось ранее, состоят из независимых ветвей, каждая из которых сама по себе укладывается в модель поведения последовательной программы, поэтому одну параллельную программу можно выполнять в нескольких процессах в псевдоодновременном режиме.

Процессы операционной системы, таким образом, являются контейнерами для многозадачного выполнения программ, как последовательных, так и параллельных.

В листинге 4.8 при помощи команды `ps` показаны процессы пользователя, упорядоченные в дерево, построенное на основе дочерне-родительских отношений между процессами.

Уникальный идентификатор, отличающий процесс от других, выведен в столбце `PID` (*process identifier*), а имя и аргументы программы, запущенной в соответствующем процессе — в столбце `COMMAND`.

В столбце STAT показано текущее состояние процесса, например S (сон, sleep) или R (выполнение, running, или готовность к выполнению, runnable):
Процессы, ожидающие завершения их операций ввода-вывода, находятся в состоянии сна, в противном случае либо выполняются, либо готовы к выполнению, т. е. ожидают, когда текущий выполняющийся процесс заснет и процессор будет переключен на них.

В столбце TIME показано чистое потребленное процессом процессорное время от момента запуска программы, увеличивающееся только при нахождении им в состоянии выполнения.

Листинг 4.8. Дерево процессов пользователя

```
fitz@ubuntu:~$ ps fx
```

PID	TTY	STAT	TIME	COMMAND
17764	tty3	Ssl+	0:00	/usr/lib/gdm3/gdm-x-session --run-script ...
17766	tty3	Sl+	3:09	_ /usr/lib/xorg/Xorg vt3 -displayfd 3 ...
17774	tty3	Sl+	0:00	_ /usr/lib/gnome-session/gnome-session-binary ...
2987	?	Ss	0:04	/lib/systemd/systemd --user
2992	?	S	0:00	_ (sd-pam)
17373	?	Ssl	0:08	_ /usr/bin/pulseaudio --daemonize=no
17444	?	Ss	0:02	_ /usr/bin/dbus-daemon --session --address=systemd: ...
17921	?	Ssl	10:04	_ /usr/bin/gnome-shell
30192	?	Ssl	0:00	_ /usr/libexec/gnome-terminal-server
30202	pts/1	-> Ss	0:00	_ bash
30226	pts/1	S+	0:00	_ man ps
30236	pts/1	S+	0:00	_ pager
30245	pts/3	-> Ss	0:00	_ bash
30251	pts/3	R+	0:00	_ ps fx
30315	?	-> Sl	0:04	_ /usr/lib/firefox/firefox -new-window
30352	?	Sl	0:02	_ /usr/lib/firefox/firefox -contentproc -childID 1 ...
30396	?	Sl	0:00	_ /usr/lib/firefox/firefox -contentproc -childID 2 ...
30442	?	Sl	0:00	_ /usr/lib/firefox/firefox -contentproc -childID 3 ...

Управляющий терминал процесса, показанный в столбце ТТУ, используется для доставки ему интерактивных сигналов .

При вводе управляющих символов intr ^C, quit ^\ и пр. У части процессов, управляющий терминал отсутствует, потому что они выполняют приложения, взаимодействующие с пользователем не посредством терминалов, а через графическую систему .

Процесс по своему определению изолирует свою программу от других выполняющихся программ, что затрудняет использование процессов для выполнения таких параллельных программ, ветви которых не являются полностью независимыми друг от друга и должны обмениваться данными.

Использование предназначенных для этого средств межпроцессного взаимодействия при интенсивном обмене приводит к обременению неоправданными накладными расходами, поэтому для эффективного выполнения таких параллельных программ используются легковесные процессы (LWP, light-weight processes), они же нити (threads).

Механизм нитей позволяет переключать центральный процессор между параллельными ветвями одной программы, размещаемыми в одном (!) процессе.

Нити никак не изолированы друг от друга, и им доступны абсолютно все ресурсы своего процесса, поэтому задача обмена данными между нитями попросту отсутствует, т. к. все данные являются для них общими.

В примере из листинга 4.9 показаны нити процесса в BSD-формате вывода.

Выбор процесса производится по его идентификатору PID, предварительно полученному командой pgrep по имени программы, выполняющейся в искомом процессе.

В выводе наличие нитей процесса отмечает флаг I (lwp) в столбце состояния STAT, а каждая строчка без идентификатора PID символизирует одну нить.

Так как в многонитевой программе переключение процессора производится между нитями, то и состояния сна S, выполнения или ожидания R приписываются отдельным нитям.

Листинг 4.9 Нити процессов, BSD-формат вывода

```
fitz@ubuntu:~$ pgrep firefox
```

```
30315
```

```
fitz@ubuntu:~$ ps mp 30315
```

PID	TTY	STAT	TIME	COMMAND
PID	TTY	STAT	TIME	COMMAND
30315	?	-	0:05	/usr/lib/firefox/firefox -new-window
-	-	•-	s1	0:03 -
-	-			...
-	-			...
-	-			...
-	-			...
-	-			...

В листинге 4.10 показаны нити процесса в SYSV-формате вывода. Выбор процесса производится по имени его программы.

Общий для всех нитей идентификатор их процесса отображается в столбце PID, уникальный идентификатор каждой нити — в столбце LWP (иногда называемый TID, thread identifier), а имя процесса (или собственное имя нити, если задано) — в столбце CMD.

Листинг 4.10. Нити процессов, SYSV-формат вывода

PID	LWP	TTY	TIME	CMD			
30315	30315	?	00:00:04	firefox			
30315	30320	?	00:00:00	gmain			
30315	30321	?	00:00:00	gdbus
30315	30328	?	00:00:00	Socket Thread
30315	30332	?	00:00:00	Cache2 I/O			
30315	30333	?	00:00:00	Cookie
30315	30371	?	00:00:00	HTML5 Parser	→		
30315	30373	?	00:00:00	DNS Resolver	#3
				

Порождение процессов и нитей, запуск программ

Несмотря на очевидные различия, историю возникновения и развития, нити в процессы объединяет общее назначение — они являются примитивами выполнена некоторого набора последовательных инструкций.

Процессы выполняют или разные последовательные программы целиком, или ветвь одной параллельной программы, но в изолированном окружении, со своим «частным» (private) набором ресурсов.

Нити, наоборот, выполняют ветви одной параллельной программы в одном окружении с «общим» (shared) набором ресурсов

В многозадачном ядре Linux вообще используется универсальное понятие «задача» которая может иметь как общие ресурсы (память, открытые файлы и т. д.) с другими задачами, так и частные ресурсы для своего собственного использования.

Порождение нового процесса (рис. 4.1, а) реализуется при помощи системного вызова `fork`, в результате которого ядро операционной системы создает новый дочерний (`child`) процесс `PID` — полную копию (`COPY`) процесса-родителя (`parent PID`).

Вся (за небольшими исключениями) память процесса — состояние, свойства атрибуты (кроме идентификатора `PID`) и даже содержимое (программа с ее библиотеками) — наследуется дочерним процессом.

Даже выполнение порожденного и порождающего процесса продолжится с одной и той же инструкции их одинаковой программы. Такое клонирование обычно используют параллельные программы с ветвями, выполняющимися в дочерних процессах.

Уничтожение процесса (например, при штатном окончании программы-) производится с помощью системного вызова `exit`. При этом родительскому процессу доставляется сигнал `SIGCHILD`, оповещающий о завершении дочернего процесса. Статус завершения `status`, переданный дочерним процессом через аргументы `exit`, будет сохраняться ядром до момента его востребования родительским процессом при помощи системного вызова `wait`, а весь этот промежуток времени дочерний процесс будет находиться в состоянии `Z` (`zombie`) (см. столбец `STAT` в листинге 4.8).

Родительский процесс может завершиться раньше своих дочерних процессов, тогда логично предположить, что все «осиротевшие» процессы окажутся зомби по завершении, потому как просто некому будет востребовать их статус завершения.

На самом деле этого не происходит, потому что «осиротевшим» процессам назначается приемный родитель, в качестве которого выступает прародитель всех процессов init с идентификатором PID = 1.

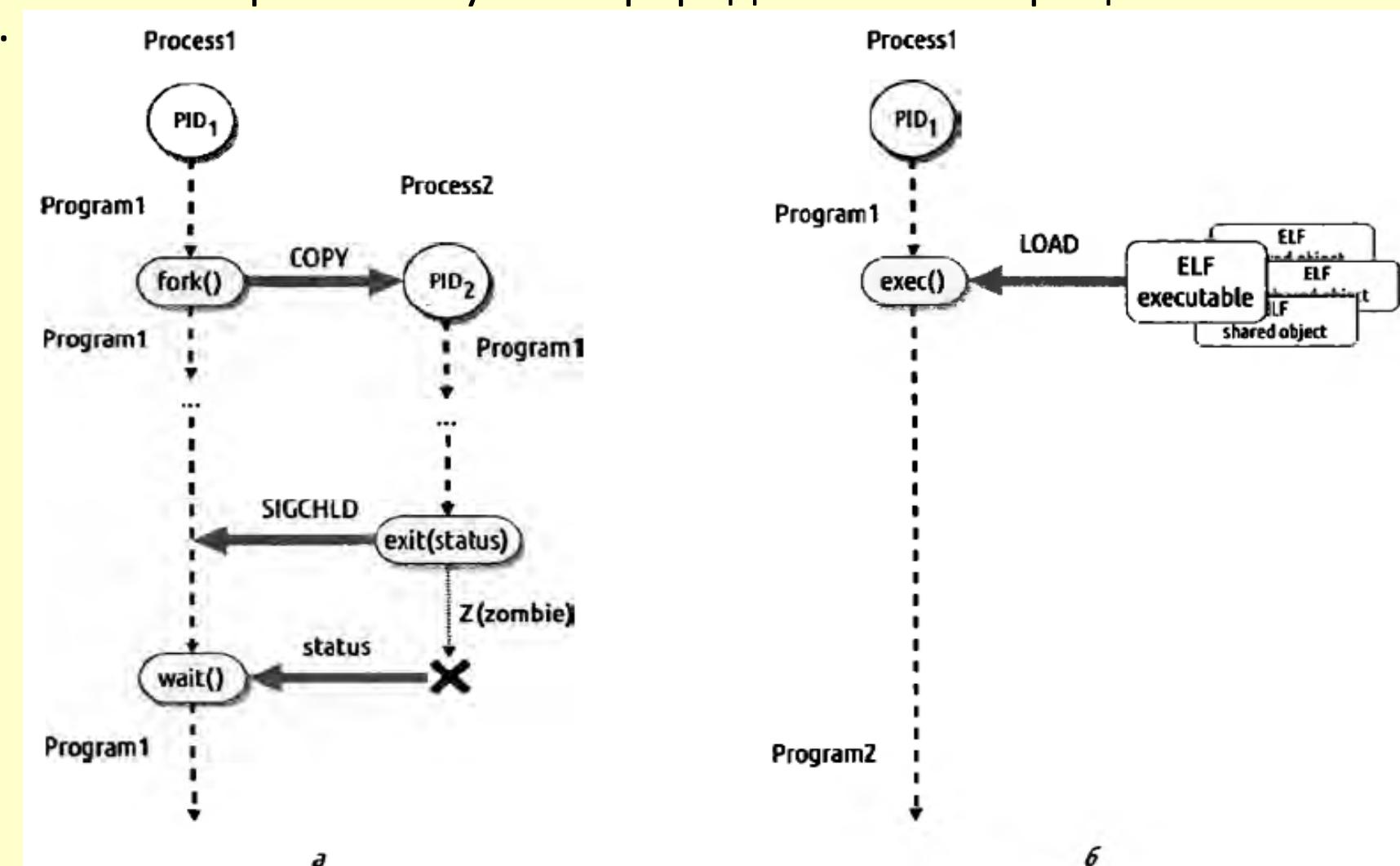


Рис. 4.1. Порождение процессов (а) и запуск программ (б)

Запуск новой программы (см. рис. 4.1, б) реализуется при помощи системного вызова exec, в результате которого содержимое процесса PID, полностью замещается запускаемой программой и библиотеками, от которых она зависит, а свойства и атрибуты (включая идентификатор PID) остаются неизменными.

Такое замещение обычно используется программами, устанавливающими нужные значения свойств и атрибутов процесса и подготавливающими ресурсы процесса к выполнению запускаемой программы.

Например, обработчик терминального доступа getty открывает заданный терминал, устанавливает режимы работы порта терминала, перенаправляет на терминал стандартные потоки ввода-вывода, а затем замещает себя программой аутентификации login.

Для запуска новой программы в новом процессе используются оба системных вызова fork и exec согласно принципу fork-and-exec «раздвоиться и запустить», показанного на рис. 4.2.

В примере из листинга 4.8 дерево процессов сформировано именно на основе дочерне-родительских отношений между процессами, формирующимиися при использовании принципа fork-and-exec.

Например, командный интерпретатор bash по командам ps fx или top ps порождает дочерние процессы и замещает их программами ps и top.

Тем же образом действует графический эмулятор терминала gnome-terminal-server — запуская новый сеанс пользователя на каждой из своих вкладок, он замещает свои дочерние процессы программой интерпретатора bash.

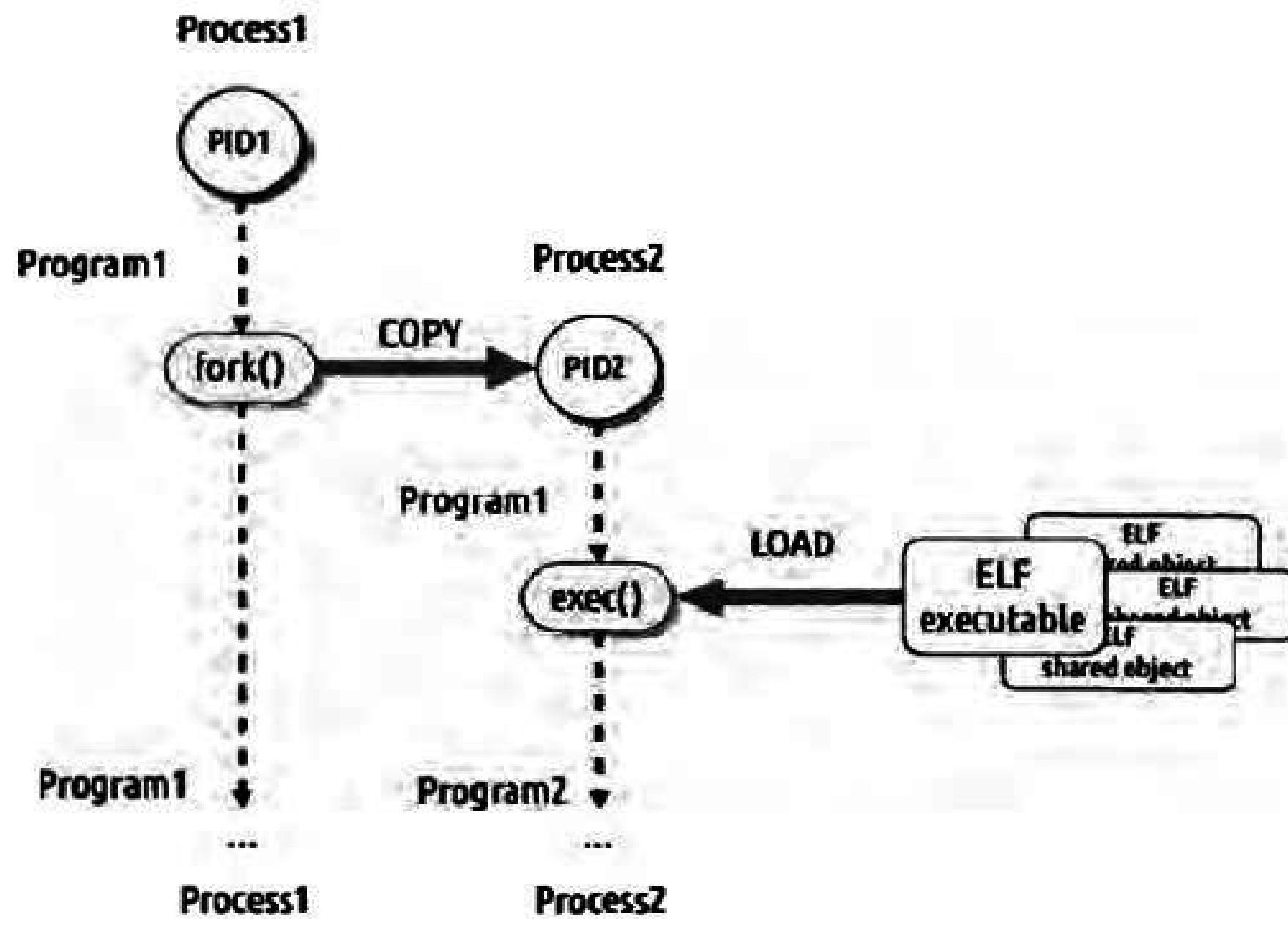


Рис. 4.2. Запуск программы в отдельном процессе

Листинг 4.11 иллюстрирует команду интерпретатора, запущенную в «фоновом» режиме при помощи конструкции асинхронного списка.

Аналогично всем предыдущим командам, интерпретатор использует fork-and-exes для запуска программы в дочернем процессе с идентификатором 23228, но не дожидается его завершения при помощи системного вызова wait, как обычно, а немедленно продолжает интерактивное взаимодействие с пользователем, сообщив ему PID порожденного процесса и «номер задания» [1] команды «заднего фона».

Оповещение о завершении своего дочернего процесса интерпретатор получит позже, при помощи сигнала SIGCHLD, и отреагирует соответствующим сообщением об окончании команды «заднего фона».

Листинг 4.11. Фоновое выполнение программ

```
fitz@ubuntu:~$ dd if=/dev/dvd of=plan9.iso
```

```
[1] 23228
```

```
fitz@ubuntu:~$ ps f
```

PID	TTY	STAT	TIME	COMMAND
23025	pts/1	S	0:00	-bash
23228	pts/1	R	1:23	_ dd if=/dev/dvd of=plan9.iso
23230	pts/1	R+	0:00	_ ps f

```
fitz@ubuntu:~$
```

```
... ... ...
```

```
fitz@ubuntu:~$ 586896+0 записей получено
```

```
586896+0 записей отправлено
```

```
300490752 байт (300 MB, 286 MiB) скопирован, 14,6916 c, 20,5 MB/c
```

```
[1]+ Завершён
```

```
dd if=/dev/dvd of=plan9.iso
```

В листинге 4.12 показана конвейерная конструкция интерпретатора, при помощи которой осуществляется поиск самого большого файла с суффиксом .html вниз по дереву каталогов, начиная с /usr/share/doc.

Эта конструкция реализуется при помощи fork-and-exes четырьмя параллельно порожденными дочерними процессами интерпретатора, в каждом из которых запущена программа соответствующей части конвейера, при этом дочерние процессы связаны неименованным каналом rpipe — простейшим средством межпроцессного взаимодействия.

Встроенная команда интерпретатора wait реализует одноименный системный вызов wait и используется для ожидания окончания всех дочерних процессов конвейера, целиком запущенного в «фоновом» режиме.

Листинг 4.12. Параллельный запуск взаимодействующих программ

```
fitz@ubuntu:~$ find /usr/share/doc -type f -name '*.html' | xargs -n1 wc -l | sort -k 1 -nr | head -1 &
          ↴          ↴          ↴          ↴          ↴
[1] 12827

fitz@ubuntu:~$ ps fj
  PPID  PID  PGID  SID TTY      TPGID STAT   UID     TIME COMMAND
11715 11716 11716  9184 pts/0    14699 S    1006  0:01 -bash
11716 12824 12824  9184 pts/0    14699 R    1006  0:00 \_ find ... -type f -name *.html
11716 12825 12824  9184 pts/0    14699 R    1006  0:00 \_ xargs -n1 wc -l
11716 12826 12824  9184 pts/0    14699 S    1006  0:00 \_ sort -k 1 -nr
11716 12827 12824  9184 pts/0    14699 S    1006  0:00 \_ head -1
11716 14699 14699  9184 pts/0    14699 R+   1006  0:00 \_ ps fj

fitz@ubuntu:~$ wait
15283 /usr/share/doc/xterm/xterm.log.html
[1]+  Завершён find /usr/share/doc -type f -name '*.html' | xargs -n1 wc -l | sort -k 1 -nr | head -1
```

Параллельные многопроцессные программы

Как указывалось ранее, параллельные программы зачастую используют процессы для выполнения отдельных ветвей.

В эту категорию часто попадают программы сетевых служб, например сервер баз данных W:[PostgreSQL], служба удаленного доступа W:[SSH] и подобные.

Листинг 4.13 иллюстрирует программу `postgres`, выполняющуюся в шести параллельных процессах, один из которых — диспетчер, четыре служебных и еще один вызван подключением пользователя `fitz` к одноименной базе данных `fitz`.

При последующих подключениях пользователей к серверу будут порождены дополнительные дочерние процессы для обслуживания их запросов — по одному на каждое подключение.

Листинг 4.13. Параллельные многопроцессные сервисы

```
fitz@ubuntu:~$ ps f -C postgres
  PID TTY      STAT   TIME COMMAND
① 6711 ?        S      0:00 /usr/lib/postgresql/11/bin/postgres -D /var/lib/postgresql...
② 6713 ?        Ss     0:00  \_ postgres: 11/main: checkpointer
| 6714 ?        Ss     0:00  \_ postgres: 11/main: background writer
| 6715 ?        Ss     0:00  \_ postgres: 11/main: walwriter
| 6716 ?        Ss     0:00  \_ postgres: 11/main: autovacuum launcher
| 6717 ?        Ss     0:00  \_ postgres: 11/main: stats collector
↳ 6718 ?        Ss     0:00  \_ postgres: 11/main: logical replication launcher
③ 9443 ?        Ss     0:00  \_ postgres: 11/main: fitz ➜ fitz [local] idle

fitz@ubuntu:~$ ssh ubuntu
fitz@ubuntu's password:
                           ...
                           ...
                           ...
Last login: Sat Nov 21 13:29:33 2015 from localhost

fitz@ubuntu:~$ ps f -C sshd
  PID TTY      STAT   TIME COMMAND
① 655 ?        Ss     0:00 /usr/sbin/sshd -D
② 21975 ?       Ss     0:00  \_ sshd: fitz [priv]
③ 22086 ?       S      0:00      \_ sshd: ➜ fitz@pts/1

fitz@ubuntu:~$ ^Dвыход
Connection to ubuntu closed.
```

Аналогично, при удаленном доступе по протоколу SSH программа `sshd`, работая в качестве диспетчера в одном процессе, на каждое подключение порождает один свой клон, который, выполнив аутентификацию и авторизацию пользователя в системе, порождает еще один свой клон, имперсонирующийся в пользователя и обслуживающий его запросы.

Параллельные многонитевые программы

Для управления нитями в Linux используют стандартный POSIX-интерфейс `pthreads`, реализующийся библиотекой W:[NPTL], которая является частью библиотеки `libc`.

Интерфейс предоставляет «нитевой» вызов создания нити `pthread_create`, который является условным аналогом «процессных» `fork` и `exec`, вызов завершения и уничтожения нити `pthread_exit`, условно аналогичный `exit`, и вызов для получения статуса завершения нити `pthreadjoin(3)`, условно аналогичный `wait`.

В качестве типичных примеров применения нитей можно привести сетевые сервисы, которые для параллельного обслуживания клиентских запросов используют нити вместо процессов. Например, WEB-сервер apache, как показано в листинге 4.14, использует два многонитевых процесса по 27 нитей в каждом, что позволяет экономить память (за счет работы всех нитей процесса с общей памятью) при обслуживании большого количества одновременных клиентских подключений.

Листинг 4.14. Параллельные многонитевые сервисы

```
fitz@ubuntu:~$ ps f -C apache2
```

PID	TTY	STAT	TIME	COMMAND
10129	?	Ss	0:00	/usr/sbin/apache2 -k start
10131	?	Sl	-	__ /usr/sbin/apache2 -k start
10132	?	Sl	-	__ /usr/sbin/apache2 -k start

```
fitz@ubuntu:~$ ps fo pid,nlwp,cmd -C apache2
```

PID	NLWP	CMD
10129	1	/usr/sbin/apache2 -k start
10131	27	__ /usr/sbin/apache2 -k start
10132	27	__ /usr/sbin/apache2 -k start

```
fitz@ubuntu:~$ ps -fLC rsyslogd
```

UID	PID	PPID	LWP	C	NLWP	S	TIME	TTY	TIME	CMD
syslog	606	1	606	0	4	ноя18	?		00:00:00	/usr/sbin/rsyslogd -n -iNONE
syslog	606	1	680	0	4	ноя18	?		00:00:00	/usr/sbin/rsyslogd -n -iNONE
syslog	606	1	681	0	4	ноя18	?		00:00:00	/usr/sbin/rsyslogd -n -iNONE
syslog	606	1	682	0	4	ноя18	?		00:00:00	/usr/sbin/rsyslogd -n -iNONE

Аналогично, сервис централизованной журнализации событий rsyslogd использует нити для параллельного сбора событийной информации из разных источников, ее обработки и журнализации.

Одна нить считывает события ядра из /proc/kmsg, вторая принимает события других служб из файлового сокета /run/systemd/journal/ syslog (/dev/log в ранних, до systemd системах), третья фильтрует поток принятых событий и записывает в журнальные файлы каталога /var/log/* и т. д.

Параллельная обработка потоков поступающих событий при помощи нитей производится с минимально возможными накладными расходами, что позволяет достигать колоссальной производительности по количеству обрабатываемых сообщений в единицу времени.

Распараллеливание используется не только для псевдоодновременного выполнения ветвей параллельной программы, но и для их настоящего одновременного выполнения несколькими центральными процессорами.

В примере из листинга 4.15 показано, как сокращается время сжатия ISO-образа файла при использовании параллельного упаковщика pbzip2 по сравнению с последовательным bzip2. Для измерения времени упаковки применяется встроенная команда интерпретатора time, при этом сначала измеряется время упаковки и время распаковки последовательным упаковщиком, а затем — время упаковки и время распаковки параллельным упаковщиком.

Команды упаковки запускаются на «заднем фоне», оценивается наличие процессов и нитей паковщиков, после чего они переводятся на «передний фон» встроенной командой интерпретатора fg (foreground) и оцениваются затраты времени.

Листинг 4.15. Параллельные многонитевые утилиты

```
fitz@ubuntu:~$ ls -lh plan9.iso
-rw-r--r-- 1 fitz fitz 287M нояб. 28 15:47 plan9.iso
fitz@ubuntu:~$ time bzip2 plan9.iso &
[1] 5545
fitz@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
 4637 pts/0    S        0:00  -bash
 5545 pts/0    S        0:00  \_ -bash
 5546 pts/0    R ->   0:12  |  \_ bzip2 plan9.iso
 5548 pts/0    R+       0:00  \_ ps f
fitz@ubuntu:~$ ps -flp 5546
UID        PID  PPID  LWP  C NLWP STIME TTY          TIME CMD
fitz      5546  5545  5546  96 -> 1 10:50 pts/0    00:00:22 bzip2 plan9.iso
fitz@ubuntu:~$ fg
time bzip2 plan9.iso
                               ⏺ ⏺ ⏺ ⏺ ⏺ ⏺ ⏺ ⏺ ⏺ ⏺ ⏺
real 0m54.780s
user 0m51.772s
sys  0m0.428s
fitz@ubuntu:~$ ls -lh plan9.iso.bz2
-rw-r--r-- 1 fitz fitz 89M нояб. 28 15:47 plan9.iso.bz2
```

```
fitz@ubuntu:~$ time bzip2 -d plan9.iso.bz2
```

① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩

real 0m20.705s ~

user 0m19.044s

sys 0m1.168s

```
fitz@ubuntu:~$ time pbzip2 plan9.iso &
```

[1] 5571

```
fitz@ubuntu:~$ ps f
```

PID	TTY	STAT	TIME	COMMAND
4637	pts/0	S	0:00	-bash
5571	pts/0	S	0:00	_ -bash
5572	pts/0	SL ~	0:03	_ pbzip2 plan9.iso
5580	pts/0	R+	0:00	_ ps f

```
fitz@ubuntu:~$ ps -flp 5572
```

UID	PID	PPID	LWP	C	NLWP	STIME	TTY		TIME	CMD
fitz	5572	5571	5578	92	-	8	10:52	pts/0	00:00:43	pbzip2 plan9.iso
				
fitz										
fitz										
fitz@ubuntu:~\$ fg										
time pbzip2 plan9.iso										

```
real 0m24.259s ~
```

```
user 1m22.940s
```

```
sys 0m1.888s
```

```
fitz@ubuntu:~$ ls -lh plan9.iso.bz2
```

```
-rw-r--r-- 1 fitz fitz 89M нояб. 28 15:47 plan9.iso.bz2
```

```
fitz@ubuntu:~$ time pbzip2 -d plan9.iso.bz2
```

```
① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩
```

```
real 0m7.384s ~
```

```
user 0m25.972s
```

```
sys 0m1.396s
```

В результате оценки оказывается, что последовательный упаковщик bzip2 использует один однопотечевой процесс и затрачивает «54,7 с реального времени на упаковку, из них «51,7 с проводит в пользовательском режиме user и лишь «0,4 с в режиме ядра sys (выполняя системные вызовы, например read или write).

Соотношение между временем режимов говорит о вычислительном характере программы, т. е. о существенном превалировании времени вычислительных операций упаковки над временем операций ввода-вывода для чтения исходных данных и записи результатов.

Это означает, что нагрузка последовательного упаковщика на центральный процессор (в случае, если бы он был единственный) близка к максимальной, и его параллельная реализация для псевдоодновременного выполнения ветвей (которые практически никогда не спят) лишена смысла.

Параллельный упаковщик pbzip2 использует один многонитевой процесс из восьми нитей и затрачивает «24,4 с реального времени на упаковку, при этом «1 мин 22,9 с (!) проводит в пользовательском режиме и «1,8 с в режиме ядра.

Прирост производительности упаковки и, как следствие, сокращение времени упаковки достигаются за счет настоящего параллельного выполнения нитей на нескольких процессорах (разных ядрах процессора).

Соотношение между реальным временем упаковки и суммарно затраченным временем режима пользователя, которое примерно в 3 раза больше, означает использование в среднем трех процессоров для параллельного выполнения вычислительных операций упаковки.

Двойственность процессов и нитей Linux

Как указывалось ранее, процессы и нити в ядре Linux сводятся к универсальному понятию «задача».

Задача, все ресурсы которой (память, открытые файлы и т. д.) используются совместно с другими такими же задачами, является нитью.

И наоборот, процессами являются такие задачи, которые обладают набором своих частных, индивидуальных ресурсов.

Универсальный системный вызов `clone` позволяет указать, какие ресурсы станут общими в порождаемой и порождающей задачах, а какие — частными. Системные вызовы порождения POSIX-процессов `fork` и POSIX-нитей `pthread_create` оказываются в Linux всего лишь «обертками» над `clone`, что проиллюстрировано в листингах 4.16 и 4.17.

В примере из листинга 4.16 архиватор `tar` PID = 11801 создает при помощи системного вызова `clone` дочерний процесс PID = 11802, в который помещает программу компрессора `gzip`, используя системный вызов `execve`.

В результате параллельной работы двух взаимодействующих процессов будет создан компрессированный архив `docs.tgz` каталога `/usr/share/doc`.

Листинг 4.16. Системный вызов clone—порождение нового процесса

```
fitz@ubuntu:~$ strace -fe clone,fork,execve tar czf docs.tgz /usr/share/doc
execve("/usr/bin/tar", ["tar", "czf", "docs.tgz", "/usr/share/doc"], ...) = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID| ... ) = 12403
tar: Удаляется начальный '/' из имен объектов
strace: Process 12403 attached
[pid 12403] execve("/bin/sh",【"/bin/sh", "-c", "gzip"], 0x7ffd8dd597c0 ...) = 0
[pid 12403] clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID| ... ) = 12404
strace: Process 12404 attached
[pid 12404] execve("/usr/bin/gzip", ["gzip"], 0x55e2e45bbb48 /* 35 vars */) = 0
    ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯
    ...
    ...
    ...
+++ exited with 0 +++
```

В примере из листинга 4.17 компрессор pbzip2 создает при помощи системного вызова clone семь «дочерних» нитей PID = 12514-42520, которые имеют общую память CLONE_VM, общие открытые файлы CLONE_FILES и прочие общие ресурсы.

Листинг 4.17 Системный вызов clone—порождение новой нити

```
fitz@ubuntu:~$ strace -fe clone,fork,execve pbzip2 plan9.iso
execve("/usr/bin/pbzip2", ["pbzip2", "plan9.iso"], 0x7ffd28884938 /* 35 vars */) = 0
clone(child_stack=0x7f1d46d38fb0, flags=CLONE_VM|CLONE_FS|CLONE_FILES|... ) = 12514
...
clone(child_stack=0x7f1d46537fb0, flags=CLONE_VM|CLONE_FS|CLONE_FILES|... ) = 12520
strace: Process 12520 attached
...
strace: Process 12514 attached
[pid 12514] +++ exited with 0 ===
...
[pid 12520] +++ exited with 0 ===
+++ exited with 0 ===
```

Дерево процессов

Процессы, попарно связанные дочерне-родительскими отношениями, формируют дерево процессов операционной системы.

Первый процесс `init`, называемый прародителем процессов, порождается ядром операционной системы после инициализации и монтирования корневой файловой системы, откуда и считывается программа `/sbin/init` (в современных системах является символической ссылкой на актуальный `/lib/systemd/systemd`).

Прародитель процессов всегда имеет PID = 1, а его основной задачей является запуск разнообразных системных служб, включая запуск обработчиков алфавитно-цифрового терминального доступа `getty`, менеджера дисплеев графического доступа, службы дистанционного доступа SSH и прочих.

Кроме того, `systemd` назначается приемным родителем для «осиротевших» процессов, а также отслеживает аварийные завершения запускаемых им служб и перезапускает их.

В примере из листинга 4.18 показано дерево процессов, построенное при помощи специальной команды `pstree`, а в листинге 4.19 — «классическое» представление дерева процессов при помощи команды `ps`.

Листинг 4.18 Дерево процессов

Процессы операционной системы принято классифицировать на системные (ядерные), демоны и прикладные, исходя из их назначения и свойств (см. листинг 4.19).

Прикладные процессы выполняют обычные пользовательские программы (например, утилиту `man`), для чего им выделяют индивидуальную память, объем которой указан в столбце `VSZ` вывода команды `ps`.

Такие процессы обычно интерактивно взаимодействуют с пользователем посредством управляемого терминала (за исключением графических программ), указанного в столбце `TTY`.

Демоны (*daemons*) выполняют системные программы, реализующие те или иные службы операционной системы. Например, `cron` реализует службу периодического выполнения заданий, `atd` — службу отложенного выполнения заданий, `rsyslogd` — службу централизованной журнализации событий, `sshd` — службу дистанционного доступа, `systemd-udevd` — службу «регистрации» подключаемых устройств, и т. д.

Демоны запускаются на ранних стадиях загрузки операционной системы и взаимодействуют с пользователем не интерактивно при помощи терминала, а опосредованно — при помощи своих утилит.

Таким образом, отсутствие управляемого терминала в столбце `TTY` отличает их от прикладных процессов.

Листинг 4.19. Процессы ядра, демоны, прикладные процессы

```
fitz@ubuntu:~$ ps faxu
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	2	0.0	0.0	0	0	?	S	ноя18	0:00	[kthreadd]
root	3	0.0	0.0	0	0	?	I<	ноя18	0:00	_ [rcu_gp]
root	4	0.0	0.0	0	0	?	I<	ноя18	0:00	_ [rcu_par_gp]
root	6	0.0	0.0	- 0	0	?	- ①	ноя18	0:00	_ [kworker/0:0H...]
root	8	0.0	0.0	0	0	?	I<	ноя18	0:00	_ [mm_percpu_wq]
root	9	0.0	0.0	0	0	?	S	ноя18	0:09	_ [ksoftirqd/0]
				
root	1	0.0	0:2	168400	11684	?	Ss	ноя18	0:12	/sbin/init splash
				
root	333	0.0	0.1	21844	5348	?	Ss	ноя18	0:07	/lib/systemd/systemd-udevd
syslog	606	0.0	0.1	224360	4244	?	Ssl	ноя18	0:01	/usr/sbin/rsyslogd -n -i...
				
root	649	0.0	0.0	20320	3036	?	- ②	Ss	ноя18	0:00 /usr/sbin/cron -f
daemon	675	0.0	0.0	3736	2184	?	Ss	ноя18	0:00 /usr/sbin/atd -f	
				
root	21545	0.0	0.0	5560	3420	tty4	Ss	ноя18	0:00 /bin/login -p --	
fitz	28152	0.0	0.0	2600	1784	tty4	S	01:38	0:00 _ -sh	
fitz	28162	0.0	0.0	12948	3584	tty4	S+	01:38	0:00 _ bash	
finn	12989	0.2	0.0	- 12092	3988	tty4	- ③	S+	13:47	0:00 _ man ps
finn	13000	0.0	0.0	10764	2544	tty4	S+	13:47	0:00 _ pager	

Системные (ядерные) процессы выполняют параллельные части ядра операционной системы, поэтому не обладают ни индивидуальной виртуальной памятью VSZ, ни управляющим терминалом ТТУ.

Более того, ядерные процессы не выполняют отдельную программу, загружаемую из ELF-файла, поэтому их имена COMMAND являются условными и изображаются в квадратных скобках, а кроме того, они имеют особое состояние I в столбце STAT.

Атрибуты процесса

Процесс в операционной системе является основным активным субъектом, взаимодействующим с окружающими его объектами — файлами и файловыми системами, другими процессами, устройствами и пр.

Возможности процесса выполнять те или иные действия по отношению к другим объектам определяются его специальными свойствами — атрибутами процесса.

Маркеры доступа

Возможности процесса по отношению к объектам, доступ к которым разграничивается при помощи дискреционных механизмов (в частности, к файлам дерева каталогов) определяются значениями его атрибутов, формирующих его DAC-маркер доступа, а именно — атрибутами RUID, RGID, EUID, EGID, см. *credentials*.

Эффективные идентификаторы EUID (effective user identifier) и EGID (effective group identifier) указывают на «эффективных» пользователя и группу, использующихся дискреционными механизмами для определения прав доступа процесса к файлам и другим объектам согласно назначенному им режиму или списку доступа.

Атрибуты RUID (real user identifier) и RGID (real group identifier) указывают на «настоящих» пользователя и группу, «управляющих» процессом.

Первому процессу пользовательского сеанса (в случае регистрации в системе с использованием алфавитно-цифрового терминала — командному интерпретатору) назначают атрибуты RUID/EUID и RGID/EGID равными идентификаторам зарегистрировавшегося пользователя и его первичной группы.

Последующие процессы пользовательского сеанса наследуют значения атрибутов, т. к. порождаются в результате клонирования при помощи `fork`. В примере из листинга 4.20 при помощи команды `id` показаны значения EUID/EGID пользовательского сеанса и их наследование от командного интерпретатора, что явным образом подтверждает команда `ps`.

Листинг 4.20. DAC-маркер доступа процесса—атрибуты RUID, EUID, RGID, EGID

```
fitz@ubuntu:~$ id  
uid=1006(fitz) gid=1008(fitz) группы=1008(fitz)  
fitz@ubuntu:~$ ps fo euid,ruid,egid,rgid,user,group,tty,cmd  


| EUID | RUID | EGID | RGID | USER | GROUP | TTY   | CMD                                |
|------|------|------|------|------|-------|-------|------------------------------------|
| 1006 | 1006 | 1008 | 1008 | fitz | fitz  | pts/2 | -bash                              |
| 1006 | 1006 | 1008 | 1008 | fitz | fitz  | pts/2 | \_ ps fo euid,uid,egid,...,tty,cmd |


```

Изменение идентификаторов EUID/EGID процесса происходит при срабатывании механизма неявной передачи полномочий, основанном на дополнительных атрибутах SUID/SGID файлов программ.

При запуске таких программ посредством системного вызова exec атрибуты EUID/EGID запускающего процесса устанавливаются равными идентификаторам UID/GID владельца запускаемой программы.

В результате процесс, в который будет загружена такая программа, будет обладать правами владельца программы, а не правами пользователя, запустившего эту программу.

В листинге 4.21 приведен типичный пример использования механизма неявной передачи полномочий при выполнении команд passwd и wall.

При смене пароля пользователем при помощи программы /usr/bin/passwd ее процесс получает необходимое право записи в файл /etc/shadow (см. листинг 3.40) в результате передачи полномочий суперпользователя root (UID=0).

При передаче широковещательного сообщения всем пользователям при помощи /usr/bin/wall необходимо иметь право записи в их файлы устройств /dev/tty*, которое появляется в результате передачи полномочий группы tty (GID = 5).

Листинг 4.21. Атрибуты файла SUID/SGID и атрибуты RUID, EUID, RGID, EGID

```
fitz@ubuntu:~$ who
fitz      pts/0          2019-11-22 00:52 (:0.0)
fitz      pts/1          2019-11-22 00:53 (:0.0)
fitz      pts/2          2019-11-22 01:06 (:0.0)

fitz@ubuntu:~$ ls -la /etc/shadow /dev/pts/*
crw--w---- 1 fitz tty    136, 2 ноя 19 12:00 /dev/pts/1
crw--w---- 1 fitz tty    136, 3 ноя 19 13:53 /dev/pts/2
c----- 1 root root    5, 2 ноя 17 03:30 /dev/pts/ptmx
-rw-r----- 1 root shadow 1647 ноя 19 12:27 /etc/shadow

fitz@ubuntu:~$ ls -l /usr/bin/passwd /usr/bin/wall
-rwsr-xr-x 1 root root 67992 авг 29 16:00 /usr/bin/passwd
-rwxr-sr-x 1 root tty 35048 авг 21 16:19 /usr/bin/wall

fitz@ubuntu:~$ ls -ln /usr/bin/passwd /usr/bin/wall
-rwsr-xr-x 1 root root 67992 авг 29 16:00 /usr/bin/passwd
-rwxr-sr-x 1 root 0 35048 авг 21 16:19 /usr/bin/wall

fitz@ubuntu:~$ ps ft pts/1,pts/2 o pid,ruid,rgid,euid,egid,tty,cmd
  PID  RUID  RGID  EUID  EGID TT      CMD
27883  1006  1008  1006  1008 pts/2  bash
27937  1006  1008  1006  * 5 pts/2  \_ wall
27124  1006  1008  1006  1008 pts/1  bash
27839  1006  1008  * 0  1008 pts/1  \_ passwd
```

По отношению к объектам, доступ к которым ограничивается при помощи мандатных механизмов, возможности процесса определяются значениями его MAC-маркера доступа, а именно — атрибутом мандатной метки LABEL. Как и RUID/EUID/RGID/EGID, атрибут LABEL назначается первому процессу сеанса пользователя явным образом, а затем наследуется при клонировании процессами-потомками от процессов-родителей.

В примере из листинга 4.22 при помощи команды id показан атрибут LABEL сеанса пользователя, а при помощи команды ps — его явное наследование от процесса-родителя.

Аналогично изменениям EUID/EGID процесса, происходящим при запуске SUID-Hofl/SGID-Hofl программы, изменение метки LABEL процесса происходит (согласно мандатным правилам) в системном вызове exec при запуске программы, помеченной соответствующей мандатной меткой файла.

Так, например, при запуске программы /usr/sbin/dhclient с типом dhcpc_exec_t ее мандатной метки процесс приобретает тип dhcpc_t своей мандатной метки, в результате чего существенно ограничивается в правах доступа к разным объектам операционной системы.

Листинг 4.22. MAC-маркер доступа процесса—мандатная метка sellinux

```
fitz@ubuntu:~$ ssh lich@fedora
lich@fedora's password:
Last login: Sat Nov 21 14:25:16 2015

[lich@centos ~]$ id -Z
staff_u:staff_r:staff_t:s0-s0:c0.c1023

[lich@centos ~]$ ps Zf
LABEL           PID TTY      STAT   TIME COMMAND
staff_u:staff_r:staff_t:s0-s0:c0.c1023 31396 pts/0 Ss   0:00 -bash
staff_u:staff_r:staff_t:s0-s0:c0.c1023 31835 pts/0 R+   0:00 \_ ps Zf
staff_u:staff_r:staff_t:s0-s0:c0.c1023 31334 tty2 Ss+  0:00 -bash
[lich@centos ~]$ sesearch -T -t dhcpc_exec_t -c process
Found 19 semantic te rules:
            ...
            ...
            ...
type_transition NetworkManager_t dhcpc_exec_t : process dhcpc_t;
            ...
            ...
            ...

[lich@centos ~]$ ls -Z /usr/sbin/dhclient
-rwxr-xr-x. root root system_u:object_r:dhcpc_exec_t:s0 /usr/sbin/dhclient

[lich@centos ~]$ ps -ZC dhclient
LABEL           PID TTY      TIME CMD
system_u:system_r:dhcpc_t:s0    2120 ?        00:00:00 dhclient
system_u:system_r:dhcpc_t:s0    4320 ?        00:00:00 dhclient
```

Привилегии

Еще одним важным атрибутом процесса, определяющим его возможности по использованию системных вызовов, являются привилегии процесса capabilities.

Например, обладание привилегией CAP_SYS_PTRACE разрешает процессам трассировщиков strace и ltrace, использующих системный вызов ptrace, трассировать процессы любых пользователей (а не только «свои», EUID которых совпадает с EUID трассировщика).

Аналогично, привилегия CAP_SYS_NICE разрешает изменять приоритет, устанавливать привязку к процессорам и назначать алгоритмы планирования процессов и нитей любых пользователей, а привилегия CAP_KILL разрешает посылать сигналы процессам любых пользователей.

Явная привилегия «владельца» CAP_FOWNER позволяет процессам изменять режим и списки доступа, мандатную метку, расширенные атрибуты и флаги любых файлов так, словно процесс выполняется от лица владельца файла.

Привилегия CAP_LINUX_IMMUTABLE разрешает управлять флагами файлов i, immutable и a, append, а привилегия CAP_SETFCAP — устанавливать «файловые» привилегии запускаемых программ.

Необходимо отметить, что именно обладание полным набором привилегий делает пользователя root (UIDsG) в Linux суперпользователем. И наоборот, обычный, непrivилегированный пользователь (в смысле UID*G) не обладает никакими явными¹ привилегиями. Назначение² привилегий процесса происходит при запуске программы при помощи системного вызова exec(3), исполняемый файл которого помечен «файловыми» привилегиями.

В примере из листинга 4.23 иллюстрируется получение списка привилегий процесса при помощи утилиты getrcaps. Как и ожидалось, процесс postgres (PID=6711), работающий от лица обычного (непrivилегированного, в смысле UID*G) псевдопользователя postgres, не имеет никаких привилегий, а процесс apache2 (PID=1G129), работающий от лица суперпользователя root (UID=0), имеет полный набор привилегий.

Однако процесс NetworkManager (PID=646) выполняется от лица “суперпользователя”, лишенного большинства своих привилегий, т. к. ему их умышленно уменьшили при его запуске) до минимально необходимого набора, достаточного для выполнения его функций.

Листинг 4.23. Привилегии (capabilities) процесса

```
fitz@ubuntu:~$ ps fo user,pid,cmd -C NetworkManager,postgres,apache2
```

USER	PID	CMD
------	-----	-----

root	10129	/usr/sbin/apache2 -k start
------	-------	----------------------------

www-data	10131	_ /usr/sbin/apache2 -k start
----------	-------	-------------------------------

www-data	10132	_ /usr/sbin/apache2 -k start
----------	-------	-------------------------------

postgres	6711	/usr/lib/postgresql/11/bin/postgres -D /var/lib/postgresql/11/main ...
----------	------	--

postgres	6713	_ postgres: 11/main: checkpointer
----------	------	------------------------------------

postgres	6714	_ postgres: 11/main: background writer
----------	------	---

postgres	6715	_ postgres: 11/main: walwriter
----------	------	---------------------------------

postgres	6716	_ postgres: 11/main: autovacuum launcher
----------	------	---

postgres	6717	_ postgres: 11/main: stats collector
----------	------	---------------------------------------

postgres	6718	_ postgres: 11/main: logical replication launcher
----------	------	--

root	646	/usr/sbin/NetworkManager --no-daemon
------	-----	--------------------------------------

Листинг 4.23. Продолжение

```
fitz@ubuntu:~$ getpcaps 6711
```

```
Capabilities for `6711': =
```

```
fitz@ubuntu:~$ getpcaps 10129
```

```
Capabilities for `10129': =
```

```
cap_chown, cap_dac_override, cap_dac_read_search, cap_fowner, cap_fsetid, cap_kill, cap_setgid,  
cap_setuid, cap_setpcap, cap_linux_immutable, cap_net_bind_service, cap_net_broadcast, cap_net_  
admin, cap_net_raw, cap_ipc_lock, cap_ipc_owner, cap_sys_module, cap_sys_rawio, cap_sys_chroot,  
cap_sys_ptrace, cap_sys_pacct, cap_sys_admin, cap_sys_boot, cap_sys_nice, cap_sys_resource, cap_  
sys_time, cap_sys_tty_config, cap_mknod, cap_lease, cap_audit_write, cap_audit_control, cap_  
setfcap, cap_mac_override, cap_mac_admin, cap_syslog, cap_wake_alarm, cap_block_suspend,  
cap_audit_read+ep
```

```
fitz@ubuntu:~$ getpcaps 646
```

```
Capabilities for `646': =
```

```
cap_dac_override, cap_kill, cap_setgid, cap_setuid, cap_net_bind_service, cap_net_admin,  
cap_net_raw, cap_sys_module, cap_sys_chroot, cap_audit_write+ep
```

В листинге 4.24 показан типичный пример применения отдельных привилегий там, где классически применяется неявная передача всех полномочий суперпользователя при помощи механизма SUID/SGID.

Например, «обычная» утилита ping для выполнения своей работы должна создать «необработанный» raw сетевой сокет, что является с точки зрения ядра привилегированной операцией.

В старых системах программа /bin/ping наделялась атрибутом SUID и находилась во владении суперпользователя root, чьи права и передавались при ее запуске.

С точки зрения защищенности системы это не соответствует здравому смыслу, подсказывающему наделять программы минимально необходимыми возможностями, достаточными для их функционирования.

Для создания «необработанных» raw и пакетных packet сокетов достаточно только привилегии CAP_NET_RAW, а весь суперпользовательский набор привилегий более чем избытен.

Листинг 4.24. Делегирование привилегий программы ping

```
fitz@ubuntu-1804:~$ ls -l /bin/ping
-rwsr-xr-x 1 root root 64424 Jun 28 11:05 /bin/ping

fitz@ubuntu-1804:~$ ping ubuntu-1804
PING ubuntu-1804 (127.0.1.1) 56(84) bytes of data.
64 bytes from ubuntu-1804 (127.0.1.1): icmp_req=1 ttl=64 time=0.074 ms
^C
--- ubuntu ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.074/0.074/0.074/0.000 ms

fitz@ubuntu-1804:~$ sudo chmod u+s /bin/ping
fitz@ubuntu-1804:~$ ls -l /bin/ping
-rwxr-sr-x 1 root root 64424 Jun 28 11:05 /bin/ping

fitz@ubuntu-1804:~$ ping ubuntu-1804
ping: icmp open socket: Operation not permitted
```

Листинг 4.24. Продолжение

```
fitz@ubuntu-1804:~$ sudo setcap cap_net_raw+ep /bin/ping
fitz@ubuntu-1804:~$ getcap /bin/ping
/bin/ping = cap_net_raw+ep
fitz@ubuntu-1804:~$ ping ubuntu-1804
PING ubuntu (127.0.1.1) 56(84) bytes of data.
64 bytes from ubuntu (127.0.1.1): icmp_req=1 ttl=64 time=0.142 ms
^C
--- ubuntu ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.142/0.142/0.142/0.000 ms
```

При отключении передачи полномочий в программа /bin/ping лишается возможности выполнять свои функции, а при назначении ей при помощи команды setcap «файловой» привилегии CAP_NET_RAW функциональность возвращается в полном объеме, т. к. приводит к установке «процессной» привилегии CAP_NET_RAW при запуске этой программы.

Для просмотра привилегий, делегируемых при запуске программ, используется парная команда getcap.

Аналогично, при использовании анализаторов сетевого трафика tshark (листинг 4.25) и/или wireshark),зывающих для захвата сетевых пакетов утилиту dumpcap, требуется открывать как «необработанные» raw, так и пакетные packet сетевые сокеты, что требует той же привилегии CAP_NET_RAW.

Классический способ применения анализаторов пакетов состоит в использовании явной передачи всех полномочий суперпользователя (при помощи su или sudo) при их запуске, что опять не соответствует минимально необходимым и достаточным требованиям к разрешенным возможностям программ.

Листинг 4.25. Делегирование привилегий программы tshark

```
fitz@ubuntu:~$ tshark
tshark: There are no interfaces on which a capture can be done
itz@ubuntu:~$ strace -fe execve tshark
execve("/usr/bin/tshark", ["tshark"], /* 23 vars */) = 0
Process 8951 attached
[pid 8951] execve("/usr/bin/dumpcap",【"/usr/bin/dumpcap", "-D", "-Z", "none"],...) = 0
Process 8951 detached
--- SIGCHLD (Child exited) @ 0 (0) ---
tshark: There are no interfaces on which a capture can be done
fitz@ubuntu:~$ ls -la /usr/bin/dumpcap
-rwxr-xr-x 1 root root 104688 Sep 5 19:43 /usr/bin/dumpcap
fitz@ubuntu:~$ getcap /usr/bin/dumpcap

fitz@ubuntu:~$ sudo setcap cap_net_raw+ep /usr/bin/dumpcap
fitz@ubuntu:~$ getcap /usr/bin/dumpcap
/usr/bin/dumpcap = cap_net_raw+ep
fitz@ubuntu:~$ tshark -i wlan0
Capturing on wlan0
0.307205 fe80::895d:9d7d:f0b3:a372 -> ff02::1:ff96:2df6 ICMPv6 86 Neighbor Solicitation
0.307460 SuperMic_74:0e:90 -> Spanning-tree-(for-bridges)_00 STP 60 Conf. Root =
32768/0/00:25:90:74:0e:90 Cost = 0 Port = 0x8001
... ...
...
```

Для эффективного использования анализаторов трафика непrivилегированными пользователями достаточно делегировать их процессам захвата пакетов привилегию CAP_NET_RAW при помощи «файловых» привилегий CAP_NET_RAW для программы захвата /usr/bin/dumpcap, что и проиллюстрировано в листинге 4.25.

Другие атрибуты

Переменные окружения (листинг 4.26) и текущий рабочий каталог (листинг 4.27) на поверку тоже оказываются атрибутами процесса, которые можно получить при помощи команд ps и pwdx соответственно.

Листинг 4.26. Переменные окружения процесса

```
fitz@ubuntu:~$ ps fe
  PID TTY      STAT   TIME COMMAND
21872 pts/2    S      0:00 -bash USER=fitz LOGNAME=fitz HOME=/home/fitz PATH=/usr/...
22904 pts/2    R+     0:00 \_ ps fe LANGUAGE=ru:ko:en LC_ADDRESS=ru_RU.UTF-8 ...
```

Листинг 4.27. Текущий рабочий каталог процесса

```
fitz@ubuntu:~$ ps fx
  PID TTY      STAT   TIME COMMAND
22984 pts/0    S      0:00 -bash
23086 pts/0    S+     0:00 \_ man ps
23097 pts/0    S+     0:00     \_ pager
21872 pts/2    S      0:00 -bash
23103 pts/2    R+     0:00 \_ ps fx
```

```
fitz@ubuntu:~$ pwdx 23097 22984
```

```
23097: /home/fitz
```

```
22984: /home/fitz
```

Классы и приоритеты процессов.

Распределение устройств ввода-вывода
между процессами.

Память процесса.

Виртуальная память.

Отображение файлов в память

Потребление памяти.

Механизм сигналов

Распределение процессора между процессами

Переключение центрального процессора между задачами (процессами и нитями) выполняет специальная компонента подсистемы управления процессами, называемая планировщиком (*scheduler*).

Именно планировщик определенным образом выбирает из множества неспящих, готовых к выполнению (*runable*) задач одну, которую переводит в состояние выполнения (*running*).

Процедуры, определяющие способ выбора и моменты выполнения выбора, называются алгоритмами планирования.

Выбор задачи, подлежащей выполнению, естественным образом происходит в моменты времени, когда текущая выполнявшаяся задача переходит в состояние сна (*sleep*) в результате выполнения операции ввода-вывода.

Вытесняющие алгоритмы планирования, кроме всего прочего, ограничивают непрерывное время выполнения задачи, принудительно прерывая ее выполнение по исчерпанию выданного ей кванта времени (*timeslice*) и вытесняя ее во множество готовых, после чего производят выбор новой задачи, подлежащей выполнению.

По умолчанию для пользовательских задач используется вытесняющий алгоритм CFS (completely fair scheduler), согласно которому процессорное время распределяется между неспящими задачами справедливым (fair) образом.

Для каждой задачи определяется выделяемая справедливая (в соответствии с ее относительным «приоритетом») доля процессорного времени, которую она должна получить при конкуренции за процессор.

Для двух задач с любыми одинаковыми приоритетами должны быть выделены равные доли (в 50% процессорного времени), а при различии в приоритетах . на одну ступень разница между выделяемыми долями должна составить «10% процессорного времени (т. е. 55 и 45% соответственно).

Для удовлетворения этого требования алгоритм планирования CFS назначает каждой ступени приоритета соответствующий вес задачи, а процессорное время делит между всеми неспящими задачами пропорционально их весам.

Таким образом, две задачи с любыми одинаковыми приоритетами будут иметь равные веса $W_i = W_j = W$, а доли процессорного времени составят $M_i = W_i(W_i + W_j) = 1/2$ и $M_j = W_j(W_i + W_j) = 1/2$.

Для двух задач с приоритетами, отличающимися на одну ступень, W_i не равно W_j , а $M_i - M_j = 1/10$, откуда несложно получить, что $W_i/W_j = 11/9$ — правило построения шкалы весов, а $M_1 = 11/20 = 0,55$ и $M_2 = 9/20 = 0,45$, что и требовалось получить.

Для дифференциации задач используют 40 относительных POSIX-приоритетов на шкале от -20 до +19, называемых «любезностью» задачи NICE.

Относительный приоритет буквально определяет, насколько «любезна» будет задача по отношению к остальным готовым к выполнению задачам при конкуренции за процессорное время освободившегося процессора.

Наименее «любезным», с относительным приоритетом -20 (наивысшим) планировщик выделит большую долю процессорного времени, а наиболее «любезным», с приоритетом +19 (наинизшим) — меньшую.

При отсутствии конкуренции, когда количество готовых к выполнению задач равно количеству свободных процессоров, приоритет не будет играть никакой роли.

В примере из листинга 4.28 при помощи команды `bzip(2)` запущены два процесса сжатия ISO-образа с наилучшим качеством одновременно друг другу на «заднем фоне».

В выводе свойств `pcpu` (percent cputime, процент потребляемого процессорного времени), `pri` (priority), `ni` (nice) и `psr` (processor number) их процессов при помощи `ps` оказывается, что они потребляют практически одинаковые доли (проценты) процессорного времени, и это для одинаковых программ вполне соответствует интуитивным ожиданиям.

После повышения любезности (понижения относительного приоритета) одного из них при помощи команды `renice` до значения + 10 отношение потребляемых долей процессорного времени не изменилось, что означает отсутствие конкуренции за процессор, подтверждаемое как столбцом PSR, показывающим номер процессора, выполняющего программу, так и командами `pgrep` и `lscpu`.

Листинг 4.28. Относительный приоритет NICE

```
fitz@ubuntu:~$ bzip2 --best -kf plan9.iso &
[1] 12944
fitz@ubuntu:~$ bzip2 --best -kf plan9.iso &
[2] 12945
fitz@ubuntu:~$ ps fo pid,pcpu,pri,ni,psr,cmd
  PID %CPU PRI NI PSR CMD
12808  0.1  19   0   0 -bash
12944 94.5  19   0   2  \_ bzip2 --best -kf plan9.iso
12945 96.0  19   0   1  \_ bzip2 --best -kf plan9.iso
12946  0.0  19   0   3  \_ ps fo pid,pcpu,pri,ni,psr,cmd

fitz@ubuntu:~$ renice +10 12945
12945 (process ID) old priority 0, new priority 10
fitz@ubuntu:~$ ps fo pid,pcpu,pri,ni,psr,cmd
  PID %CPU PRI NI PSR CMD
12808  0.1  19   0   0 -bash
12944 94.8  19   0  -2  \_ bzip2 --best -kf plan9.iso
12945 97.0   9  10  -0  \_ bzip2 --best -kf plan9.iso
12948  0.0  19   0   1  \_ ps fo pid,pcpu,pri,ni,psr,cmd
fitz@ubuntu:~$ nproc
4
fitz@ubuntu:~$ lscpu
Архитектура:                               x86_64
CPU op-mode(s):                            32-bit, 64-bit
```

Листинг 4.28. Продолжение 1

Порядок байт: Little Endian

Address sizes: 36 bits physical, 48 bits virtual

CPU(s): 4 ↗

On-line CPU(s) list: 0-3

...

fitz@ubuntu:~\$ taskset -p -c 3 12808

pid 12808's current affinity list: 0-3

pid 12808's new affinity list: 3

fitz@ubuntu:~\$ nice -n 5 time bzip2 --best -kf plan9.iso &

[1] 29331

fitz@ubuntu:~\$ nice -n 15 time bzip2 --best -kf plan9.iso &

[2] 29333

fitz@ubuntu:~\$ ps fo pid,pcpu,pri,ni,psr,cmd

PID	%CPU	PRI	NI	PSR	CMD
-----	------	-----	----	-----	-----

28573	0.0	19	0	3	-bash
-------	-----	----	---	---	-------

29331	0.0	9	5	3	_ time bzip2 --best -kf plan9.iso
-------	-----	---	---	---	------------------------------------

29332	91.4	9	5	3	_ bzip2 --best -kf plan9.iso
-------	------	---	---	---	-------------------------------

29333	0.0	4	15	3	_ time bzip2 --best -kf plan9.iso
-------	-----	---	----	---	------------------------------------

29334	9.7	4	15	3	_ bzip2 --best -kf plan9.iso
-------	-----	---	----	---	-------------------------------

29336	0.0	19	0	3	_ ps fo pid,pcpu,pri,ni,psr,cmd
-------	-----	----	---	---	----------------------------------

Листинг 4,28. Продолжение 2

```
fitz@ubuntu:~$ wait  
51.10user 0.22system 0:56.86elapsed 99%CPU (0avgtext+0avgdata 31248maxresident)k  
0inputs+180560outputs (0major+1004minor)pagefaults 0swaps  
[1]- Завершён           nice -n 5 time bzip2 --best -kf plan9.iso  
53.79user 0.20system 1:43.08elapsed 52%CPU (0avgtext+0avgdata 31520maxresident)k  
0inputs+180560outputs (0major+1515minor)pagefaults 0swaps  
[2]+ Завершён           nice -n 15 time bzip2 --best -kf plan9.iso
```

Проиллюстрировать действие относительного приоритета NICE на многопроцессорной системе можно, создав искусственную конкуренцию двух процессов за один процессор.

Для этого при помощи команды taskset устанавливается привязка (affinity) командного интерпретатора (PID = 12808) к процессору 3 (привязка, как и прочие свойства и атрибуты процесса, наследуется потомками).

Затем при помощи команды nice запускаются две программы упаковки с относительными приоритетами 5 и 15 в режиме измерения потребления времени при помощи команды time.

В результате доли процессорного времени распределяются неравномерно, причем их разница зависит от разницы в относительных приоритетах (и от свойств конкурирующих процессов, но в примере они одинаковые).

Дождавшись завершения процессов заднего фона, при помощи встроенной команды wait можно оценить разницу в реальном времени выполнения упаковки, вызванную неравным распределением процессора между процессами упаковщиков.

Кроме приоритетной очереди, планировщик Linux позволяет использовать еще три алгоритма планирования — FIFO, RR и EDF, предназначенные для задач реального времени.

Вытесняющий алгоритм RR (round robin) организует простейшее циклическое обслуживание с фиксированными квантами времени, тогда как FIFO (first in first out) является его невытесняющей модификацией, позволяя задаче выполнять непрерывно долго, до момента ее засыпания.

По сути, оба алгоритма организуют задачи в одну приоритетную очередь (PQ, priority queue) со статическими приоритетами на шкале от 1 до 99, выбирая для выполнения всегда самую высокоприоритетную из множества готовых.

Алгоритм EDF (Earliest Deadline First) предназначен для обеспечения гарантий периодическим задачам реального времени, которым важно получать периодическое обслуживание так, чтобы задача не была вытеснена в течение определенного времени.

Перевод задачи под управление тем или иным алгоритмом планирования производится при помощи назначения ей политики планирования (scheduling policy) посредством команды chrt.

Различают шесть политик планирования, три из которых — SCHED_OTHER, SCHED_BATCH и SCHED_DEADLINE, реализуются алгоритмом CFS, политики SCHED_FIFO, SCHED_RR — одноименными алгоритмами FIFO и RR, а политика SCHED_DEADLINE — алгоритмом EDF.

Политика SCHED_OTHER, она же SCHED_NORMAL, применяется по умолчанию и обслуживает класс задач TS (time sharing), требующих «интерактивности», а политика SCHED_BATCH предназначается для выполнения «вычислительных» задач класса пакетной В (batch) обработки.

Разница между политиками состоит в том, что планировщик в случае необходимости всегда прерывает задачи класса В в пользу задач класса TS, но никогда наоборот.

Политика SCHED_IDLE формирует класс задач, выполняющихся только при «простое» (idle) центрального процессора за счет выделения им планировщиком CFS очень небольшой доли процессорного времени.

Для этого задачам IDL-класса назначают минимально возможный вес — меньший, чем вес самых «любезных» (NICE = +19) задач TS -класса.

В примере из листинга 4.29 проиллюстрировано распределение процессорного времени алгоритмом планирования CFS между (конкурирующими за один процессор) одинаковыми процессами TS-, B- и IDL-классов, назначенных им при запуске упаковщиков bzip2 посредством команды chrt.

Листинг 4.29. Классы процессов

```
fitz@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
12058 pts/0    Ss      0:00 bash
12065 pts/0    R+      0:00  \_ ps f
fitz@ubuntu:~$ taskset -p -c 3 12058
pid 12058's current affinity list: 0-3
pid 12058's new affinity list: 3

fitz@ubuntu:~$ chrt -b 0 time bzip2 --best -kf plan9.iso &
[1] 12410
fitz@ubuntu:~$ chrt -o 0 time bzip2 --best -kf plan9.iso &
[2] 12412
fitz@ubuntu:~$ chrt -i 0 time bzip2 --best -kf plan9.iso &
[3] 12414
```

Листинг 4.29. Продолжение

```
fitz@ubuntu:~$ ps fo pid,pcpu,class,pri,ni,psr,cmd
  PID %CPU CLS   PRI  NI PSR CMD
12058 0.1 TS     19   0   3 -bash
12410 0.0 B     19   -   3  \_ time bzip2 --best -kf plan9.iso
12411 50.0 B    -19   -   3  |  \_ bzip2 --best -kf plan9.iso
12412 0.0 TS     19   0   3  \_ time bzip2 --best -kf plan9.iso
12413 49.7 TS    -19   0   3  |  \_ bzip2 --best -kf plan9.iso
12414 0.0 IDL    19   -   3  \_ time bzip2 --best -kf plan9.iso
12415 0.1 IDL   -19   -   3  |  \_ bzip2 --best -kf plan9.iso
12471 0.0 TS     19   0   3  \_ ps fo pid,pcpu,class,pri,ni,psr,cmd
fitz@ubuntu:~$ wait
53.85user 0.26system 1:45.98elapsed 51%CPU (0avgtext+0avgdata 31248maxresident)k
0inputs+180560outputs (0major+1004minor)pagefaults 0swaps
53.96user 0.22system 1:46.04elapsed 51%CPU (0avgtext+0avgdata 31248maxresident)k
0inputs+180560outputs (0major+1004minor)pagefaults 0swaps
52.74user 0.27system 2:41.54elapsed 32%CPU (0avgtext+0avgdata 31248maxresident)k
0inputs+180560outputs (0major+1004minor)pagefaults 0swaps
[1]  Завершён          chrt -b 0 time bzip2 --best -kf plan9.iso
[2]- Завершён          chrt -o 0 time bzip2 --best -kf plan9.iso
[3]+ Завершён          chrt -i 0 time bzip2 --best -kf plan9.iso
```

В листинге 4.30 показана конкуренция процессов под управлением RR-планировщика, использующего статические приоритеты.

Так как операция назначения политик планирования «реального времени» FIFO и RR является привилегированной, то сначала командный интерпретатор переводится в RR-класс (-г) с наивысшим статическим приоритетом 99 при помощи команды chrt, выполняемой от лица суперпользователя root.

При последующих запусках упаковщиков класс будет унаследован и не потребует повышенных привилегий.

Два процесса упаковщиков запускаются командой chrtc одинаковыми статическими приоритетами 1, привязанные одному процессору командой taskset, в результате чего получают равные доли процессорного времени, что вполне соответствует интуитивным ожиданиям от вытесняющего циклического планировщика RR.

Нужно отметить, что шкала статических приоритетов 1->99 классов RR и FIFO, как и шкала «любезности» NICE +19->-20 классов TS и B, отображаются на общую шкалу приоритетов PRI так, что верхняя часть шкалы PRI:41->139 соответствует статическим приоритетам, а нижняя часть шкалы PRI:0->39 соответствует «любезности».

Листинг 4.30. Классы процессов реального времени

```
fitz@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
15313 pts/0    S      0:00 -bash
15520 pts/0    R+     0:00 \_ ps f
fitz@ubuntu:~$ sudo chrt -pr 99 15313
fitz@ubuntu:~$ ps fo pid,psr,cls,ni,pri,pcpu,comm
  PID PSR CLS NI PRI %CPU COMMAND
15313  0  RR   - 139  0.1 bash
15550  1  RR   - 139  0.0 \_ ps

fitz@ubuntu:~$ chrt -r 1 taskset -c 2 bzip2 --best -kf plan9.iso &
[1] 15572
fitz@ubuntu:~$ chrt -r 1 taskset -c 2 bzip2 --best -kf plan9.iso &
[2] 15573
fitz@ubuntu:~$ ps fo pid,psr,cls,ni,pri,pcpu,comm
  PID PSR CLS NI PRI %CPU COMMAND
15313  0  RR   - 139  0.1 bash
15572  2  RR   - 41  51.8 \_ bzip2
★15573  2  RR   - 41  48.8 \_ bzip2
15597  1  RR   - 139  0.0 \_ ps
```

Листинг 4.30. Продолжение

```
fitz@ubuntu:~$ chrt -r 2 taskset -c 2 bzip2 --best -kf plan9.iso &
```

```
[3] 15628
```

```
fitz@ubuntu:~$ ps fo pid,psr,cls,ni,pri,pcpu,comm
```

PID	PSR	CLS	NI	PRI	%CPU	COMMAND
15313	0	RR	-	139	0.1	bash
15572	2	RR	-	41	48.3	*? _ bzip2
15573	2	RR	-	41	47.5	*? _ bzip2
15628	2	RR	-	42	93.3	*! _ bzip2
15630	1	RR	-	139	0.0	_ ps

```
fitz@ubuntu:~$ top -b -n1 -p 15572,15573,15628
```

```
top - 14:44:01 up 4:27, 2 users, load average: 5.07, 3.42, 2.50
```

```
Tasks: 3 total, 3 running, 0 sleeping, 0 stopped, 0 zombie
```

```
Cpu(s): 18.5%us, 3.4%sy, 0.6%ni, 76.0%id, 1.4%wa, 0.0%hi, 0.1%si, 0.0%st
```

```
МиБ Mem : 3935,6 total, 2629,5 free, 425,1 used, 880,9 buff/cache
```

```
МиБ Swap: 448,5 total, 448,5 free, 0,0 used. 3265,3 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
15628	fitz	-	-3	0	9536	7880	468 R	100	0.1	1:14.96	bzip2
15572	fitz	-	-2	0	9536	7880	460 R	0	0.1	1:30.95	bzip2
15573	fitz	-	-2	0	9536	7884	464 R	0	0.1	1:30.01	bzip2

Добавление третьего процесса упаковщика со статическим приоритетом 2 приводит к резкому перекосу выделяемой доли процессорного времени в его пользу.

Это объясняется тем, что алгоритм планирования RR (равно как и FIFO) всегда выбирает процесс с самым высоким статическим приоритетом из множества готовых, поэтому процессам с более низкими приоритетами процессорное время будет выделено только при засыпании всех процессов с большими приоритетами.

Странный результат ★, изображаемый командой ps, объясняется «несовершенством» ее способа расчета доли процессорного времени %CPU, выделяемой процессу.

Расчет производится как отношение чистого потребленного процессорного времени (за все время существования процесса) к промежутку реального времени, прошедшему с момента порождения процесса, что соответствует среднему, но не мгновенному значению потребляемой доли.

Гораздо более ожидаемый результат получает команда top, выполняющая расчет мгновенной доли процессорного времени как отношение чистого потребленного процессорного времени (за небольшой промежуток наблюдения) к реальному времени наблюдения.

Распределение устройств ввода-вывода между процессами

В любой многозадачной операционной системе кроме вопроса распределения (между ее задачами) центрального процессора, рассмотренного выше, неизбежно возникают вопросы распределения и других устройств, например памяти и устройств ввода-вывода.

В Linux все устройства ввода-вывода принято подразделять на «поблочные» устройства (block devices), которые зачастую являются накопителями, «посимвольные» устройства (character devices), - как правило, устройства взаимодействия с пользователями (мыши, клавиатуры, терминалы и пр.) и «попакетные» устройства, в большинстве своем — сетевые интерфейсы.

В каждом классе устройств способы их распределения часто связаны с эффективностью их работы, т. к. решение задачи «в лоб» приводит к неприемлемым результатам.

Например, при распределении доступа к дисковым накопителям оказывается, что количество считываемых данных в единицу времени (пропускная способность, throughput) существенно зависит от того, в каком порядке обслуживать запросы отдельных процессов на чтение/запись накопителя.

Такой результат в основном обусловлен свойствами самих накопителей, все еще являющихся механическими дисками.

Для проведения операции чтения или записи дискового блока контроллер накопителя должен потратить время на перемещение головки над нужной дорожкой, а затем дождаться «приезда» нужного сектора этой дорожки, который содержит данные искомого блока.

На все это нужно колоссальное (с точки зрения центрального процессора и процессов) время, что и ограничивает количество данных, обрабатываемых в единицу времени.

Если представить себе операции линейного считывания или записи, когда обрабатываются дисковые блоки, находящиеся в последовательных секторах одной дорожки, затем последовательно в секторах соседней дорожки, то можно добиться максимальной производительности, но это невозможно по двум причинам.

Во-первых, процессы работают с абстракциями более высокого уровня — с файлами, данные которых могут размещаться файловыми системами в произвольных дисковых блоках.

Даже если последовательные блоки файла размещены в последовательных дисковых блоках, программы в принципе могут читать произвольные файлы в каком угодно порядке.

Во-вторых, сама мультипрограммная смесь в принципе генерирует общий поток запросов к произвольным местам диска.

При попытке обслуживать этот поток «как есть» количество накладных расходов на хаотичный поиск (seek time) нужных дисковых блоков будет достаточно велико, а результирующая пропускная способность диска — мала.

Именно эта задача приводит к появлению в подсистеме блочных устройств ядра планировщика ввода-вывода (I/O scheduler), т. е. специальной компоненты, обслуживающей очередь запросов к накопителю особым образом, изначально направленным на оптимизацию пропускной способности накопителя.

Подобные планировщики часто используют алгоритмы, подобные тем, что задействованы в лифтовых системах зданий для оптимизации перемещения лифтов (elevator), и носят название лифтовых алгоритмов.

Наиболее известными алгоритмами являются так называемые SCAN, C-SCAN, LOOK и C-LOOK, см. W:[Elevator algorithm], W:[LOOK algotithm].

Принцип их действия (упрощенно) состоит в том, что начавший движение лифт всегда едет в одном направлении, подбирая новых пассажиров, только направляющихся в сторону его движения, после чего, достигнув точки назначения, ждет нового вызова.

Планировщики ввода-вывода аналогично пытаются обслуживать поступающие запросы к накопителю не в порядке их поступления, а в порядке их номеров блоков (в порядке номеров этажей, лифт всегда едет снизу вверх), что естественно уменьшает суммарное время поиска и, как следствие, увеличивает общую пропускную способность.

Кроме этого, планировщики пытаются укрупнять мелкие запросы на доступ к единичным, расположенным последовательно дисковым блокам, объединяя их в меньшее количество крупных запросов на доступ к «несколько блочным» дисковым областям.

Это опять позволяет экономить время при обращении к диску, что тоже увеличивает количество данных, обрабатываемых в единицу времени.

Две такие применяемые планировщиками операции принято называть сортировкой (sorting) и слиянием (merging) соответственно.

Простейший планировщик в стареньких ядрах версии 2.4 организовывал общую очередь запросов к накопителю, отсортированную в порядке их номеров блоков, помещая новые запросы в нужное место в середине очереди, а обслуживал запросы всегда из головы очереди, отправляя их на обработку драйверу контроллера накопителя.

Организованная таким образом обработка, однако, должна страдать от эффекта голодания (*starvation*) запросов со старшими номерами, т. к. если представить, что новые запросы будут непрерывно и достаточно интенсивно поступать с младшими номерами блоков, то запросы со старшими номерами вообще никогда не будут обработаны.

Для решения проблемы голодания этот классический планировщик использовал возрастные отметки запросов, заставляя новые запросы размещаться всегда в конце очереди, если в ней был найден хотя бы один запрос старше определенного возраста. Такая эвристика уменьшала проблему, но в принципе не избавляла от нее, т. к. запросы, конечно, не застревали в очереди «навсегда», но задержка их обработки была непредсказуемой и ожидала желаТЬ лучшего.

Более того, она никак не касалась особенного случая эффекта голодания — так называемого голодания запросов на чтение, вызываемого запросами на запись (*writes starving reads*).

Этот эффект основывается на том, что при чтении в подавляющем большинстве случаев по тем или иным причинам программы ждут реального завершения одного запроса, прежде чем отправят другой.

При записи, наоборот, они практически всегда генерируют кучу запросов пачкой, отчасти и потому, что ядро воображаемо «завершает» для процесса операцию записи немедленно после копирования данных запроса в свои внутренние структуры (например, в страницочный кэш).

Таким образом, в очереди запросов оказывается куча запросов на запись, обработка которых неминуемо ведет к задержкам обработки немногих запросов на чтение.

Ситуация еще больше усугубляется тем, что подобные пачки запросов на запись обычно адресуют последовательные блоки, что, к сожалению (для эффекта голодания), согласуется с лифтовыми алгоритмами классического планировщика.

Одним из первых планировщиков ввода-вывода, который был направлен на решение вышеобозначенных проблем, стал появившийся в ядрах версии 2.6 так называемый **deadline**, доступный и по сей день.

Принцип использования «лифтовых» алгоритмов остался без изменения, и в планировщике также организуется общая очередь запросов, отсортированная в порядке их номеров дисковых блоков.

Однако для решения проблем голодания планировщик организует еще две отдельные очереди — отдельную для запросов на чтение и отдельную для запросов на запись, в которых запросы размещены в порядке поступления.

Более того, у каждой из дополнительных очередей есть определенные преследуемые «сроки» обработки, определенные в пользу запросов на чтение.

По умолчанию они составляют 500 мс для чтения и 5000 мс (5 с) для запросов на запись.

В основном режиме планировщик обслуживает запросы в порядке общей очереди, что направлено на повышение пропускной способности накопителя, но по истечении определенного «срока» обработки запросов в одной из дополнительных очередей планировщик переходит в режим обработки этой дополнительной очереди, тем самым пытаясь обеспечить обозначенные «сроки».

Несмотря на свою простоту, **deadline**-планировщик справился с эффектами голодания, обеспечив вполне предсказуемые значения задержек обработки запросов.

Однако надо заметить, что deadline-планировщик делает свою работу за счет уменьшения общей пропускной способности, т. к. каждый раз при истечении срока обработки запроса на чтение он прерывает обслуживание в «лифтовом» порядке и тратит дополнительное время на поиск блока этого истекшего запроса, после чего возвращается к обычной работе и опять тратит дополнительное время на поиск того блока, на котором он прервался.

Такие наблюдения привели к эвристике «предвосхищающего» anticipatory-планировщика, который основывается на поведении deadline-планировщика и предвидении запросов на чтение, следующих за обработанными запросами с истекшим «сроком».

Другими словами, каждый раз, когда anticipatory-планировщик отвлекается на запрос чтения с истекшим «сроком», по окончании его обслуживания он не спешит возвращаться к основной очереди, а выжидает некоторое непродолжительное время (6 мс по умолчанию) в надежде на получение еще одного запроса на чтение с номером блока, следующим за только что обработанным.

В силу широко распространенного последовательного чтения из файлов такое предвидение оправдывается с высокой вероятностью, что приводит к повышению пропускной способности при сохранении предсказуемых задержек операций чтения.

Впоследствии anticipatory-планировщик был удален из ядра в пользу планировщика CFQ (Completely Fair Queuing, совершенно справедливая очередь), который позволяет получать результаты сравнимые (и даже лучше) с anticipatory-планировщиком, но обладает еще и массой других полезных свойств.

В основе CFQ-планировщика лежит желание обеспечить справедливое распределение пропускной способности накопителя между процессами, вне зависимости от их поведения.

Для этого планировщик организует для запросов на чтение по одной очереди на процесс, плюс общую очередь для запросов на запись, а сами запросы во всех очередях, как и всегда, сортируются в порядке номеров их дисковых блоков.

Кроме этого, подобно deadline-планировщику, CFQ определяет максимальные «сроки» обработки запросов, а подобно anticipatory-планировщику, после обработки запроса на чтение выжидает некоторое время, предвосхищая появление еще одного запроса с номером блока, следующего за только что обработанным.

Кроме этого, планировщик предусматривает возможность дифференциации процессов (и, как следствие, их очередей чтения) по классам и приоритетам, позволяя выделить долю пропускной способности для определенных процессов чуть «справедливее», чем для других.

Очереди записи для всех процессов общие, но тоже подразделяются по классам и приоритетам.

Порядок выбора очереди для обработки запросов определяется сначала ее классом, а внутри класса — при помощи специальных отметок «времени начала» обработки, назначаемых в зависимости от приоритета.

При этом запросы в каждой из них обрабатываются в течение времени, ограниченного сверху некоторым интервалом (slice) времени, так же определяемым приоритетом очереди.

Различают три класса обслуживания: realtime, best-effort, idle, которые задаются процессам явно при помощи системного вызова `ioprio_set` и утилиты `ionice`.

В случае, если для процесса явно не определен класс обслуживания (по умолчанию все процессы отнесены к «классу» `none`), он неявно «зеркалируется» на приоритет планировщика центрального процессора CFS.

Так, например, процессы из CFS класса `SCHED_DEADLINE` неявно расцениваются как находящиеся в CFQ -классе `idle`, а процессы из классов `SCHED_FIFO`, `SCHED_RR` и `SCHED_DEADLINE` — как в классе `realtime`.

Для остальных процессов используется класс `best-effort`, при этом не заданные явно приоритеты так же неявно высчитываются пропорционально «любезности» `NICE`.

В классах `realtime` и `best-effort` различают по 8 приоритетов (min 7-->0 max), от которых зависит длительность интервала времени, выделяемого очередям на обработку.

Базовый интервал времени S обычно равняется 100 мс (см. в листинге 4.32), который назначается «среднему» приоритету $p = 4$, а для остальных они масштабируются от 40 мс (для приоритета 7) до 180 мс (для приоритета 0), как $S_p = S + S * k * (4 - p)$, где $k = 1/5$ — масштабный множитель шкалы приоритетов.

Для обработки запросов на запись базовый интервал составляет всего лишь 40 мс (см. в листинге 4.32).

Кроме того, приоритеты так же используются для определения «времени начала» обработки запросов каждой очереди (*slice offset*), как $SO_p = (n - 1) * (S_0 - S_p)$, где n — количество непустых очередей планировщика.

Например, если есть три активно читающих процесса $n = 3$, то для процессов с приоритетом 0 время начала обработки определяется как $SO_0 = 0$ мс, а для процессов с приоритетом 7 - как $SO_7 = 2 * (180 - 40) = 360$ мс соответственно.

Запросы в очередях класса `realtime` обрабатываются в первую очередь, и если все запросы исчерпаны, то планировщик переходит к обработке `best-effort`-очередей, а при их исчерпании — к очередям класса `idle`.

Внутри каждого класса очереди обрабатываются в порядке «времени начала» обработки и в течение интервала обработки, а сами запросы — в порядке номеров их дисковых блоков.

Вместе с тем, как было сказано раньше, CFQ-планировщик попроцессно отслеживает максимально установленные «сроки» обработки запросов, что составляет 250 мс для запросов записи и 125 мс для запросов чтения (см. в листинге 4.32).

При исчерпании запросов в обрабатываемой очереди (если еще не истек ее интервал) CFQ-планировщик выжидает 8 мс (см. в листинге 4.32), предвидя появление нового запроса на чтение, «продолжающего» только что обработанный.

Внутреннее устройство CFQ-планировщика, надо заметить, изначально не было направлено (в отличие от «классического», `deadline` и `anticipatory`) на увеличение пропускной способности накопителя, а преследовало несколько иные цели.

Однако его современная реализация оказалась весьма эффективной для самых разнообразных применений, что и сделало его планировщиком по умолчанию вплоть до ядер версии 5.x (см. листинги 4.31 и 4.32), где он был заменен планировщиком BFQ (Budget Fair Queue).

Листинг 4.31. I/O планировщики устройств (single-queue)

```
fitz@ubuntu:~$ lsblk -S
```

NAME	HCTL	TYPE	VENDOR	MODEL	REV	TRAN
sda	0:0:0:0	disk	ATA	WDC WD2500BKT-7	1A01	sata
sr0	1:0:0:0	rom	TSSTcorp	DVD+-RW TS-U633J	D600	sata

```
fitz@ubuntu:~$ cat /sys/block/{sda,sr0}/queue/scheduler
```

```
noop deadline [cfq] ↳  
noop deadline [cfq]
```

```
fitz@ubuntu:~$ uname -r
```

```
4.18.0-25-lowlatency
```

```
fitz@ubuntu:~$ echo deadline | sudo tee /sys/block/sr0/queue/scheduler  
deadline
```

```
fitz@ubuntu:~$ cat /sys/block/sr0/queue/scheduler  
noop [deadline] ↳ cfq
```

Листинг 4.31 показывает, что назначенный блочному устройству планировщик можно узнать только при помощи «прямого» чтения файловой системы `sysfs`, а выбрать иной планировщик при помощи «прямой» записи в ее файлы.

Нужно отметить, что планировщик `noop` (по ореартион), как можно догадаться из названия, (почти) ничего с поступающими запросами не делает, но именно поэтому идеально подходит для недисковых устройств, например SSD-накопителей, задержки доступа к данным которых никак не определяются механической составляющей, присущей «обычным», дисковым накопителям.

Листинг 4.32 Параметры I/O планировщика CFQ

```
fitz@ubuntu:~$ ls /sys/block/sda/queue/iосched/
```

```
back_seek_max      group_idle_us    slice_async_us  target_latency
```

```
back_seek_penalty  low_latency     slice_idle       target_latency_us
```

```
fifo_expire_async  quantum        slice_idle_us
```

```
fifo_expire_sync   slice_async     slice_sync
```

```
group_idle         slice_async_rq  slice_sync_us
```

```
fitz@ubuntu:~$ cat /sys/block/sda/queue/iосched/slice_sync
```

```
100
```

```
fitz@ubuntu:~$ cat /sys/block/sda/queue/iосched/slice_async
```

```
40
```

```
fitz@ubuntu:~$ cat /sys/block/sda/queue/iосched/fifo_expire_*
```

```
250
```

```
125
```

```
fitz@ubuntu:~$ cat /sys/block/sda/queue/iосched/slice_idle
```

```
8
```

Широкое распространение сверхбыстрых твердотельных накопителей, например W:[NVMe], поставило перед разработчиками ядра Linux новые задачи, т. к. оказалось, что подсистема блочных устройств в силу внутреннего строения в принципе не способна обрабатывать большое количество запросов на чтение/запись в единицу времени W: [IOPS], что не позволяет использовать такие накопители на «полную катушку».

При разработке ядер серии 4.x блочную систему перепроектировали и изменили принцип организации запросов, поступающих к блочному устройству на обработку.

Вместо одной общей входящей очереди (single queue) к каждому устройству, за которую ранее состязались процессы, выполняющиеся на разных процессорах, организовали индивидуальные очереди (multi-queue) по числу процессоров, что позволило существенно повысить эффективность работы подсистемы.

Это привело к переделке драйверов устройств и планировщиков ввода-вывода к новому виду, которая закончилась к началу ядер серии 5.x.

Вместе с тем планировщик deadline превратился в mq-deadline, место CFQ занял BFQ, а позднее был добавлен планировщик kyber (листинг 4.33) для работы с высокоскоростными накопителями.

Листинг 4.33 I/O планировщики устройств (multi-queue)

```
fitz@ubuntu:~$ cat /sys/block/{sda,sr0}/queue/scheduler
mq-deadline [bfq] none
[mq-deadline] bfq none

fitz@ubuntu:~$ uname -r
5.3.0-24-lowlatency

fitz@ubuntu:~$ modinfo bfq
filename:      /lib/modules/5.3.0-26-generic/kernel/block/bfq.ko
description:   MQ Budget Fair Queueing I/O Scheduler
               ...
               ...
               ...

fitz@ubuntu:~$ modinfo kyber-iosched
filename:      /lib/modules/5.3.0-26-generic/kernel/block/kyber-iosched.ko
description:   Kyber I/O scheduler
               ...
               ...
               ...
```

Планировщик BFQ изначально базировался на коде CFQ и до сих пор сохраняет практически все его свойства, но вместо интервалов времени, отводимых на обработку запросов, использует понятие «бюджета» обработки, исчисляемого в количестве (дисковых) секторов, которые будут обслужены из той или иной очереди.

Бюджет BFQ(как и интервалы обработки CFQ) пропорционален приоритетам процессов, что делает планировщик «справедливым» в отношении доли пропускной способности устройства, выделяемой отдельным процессам.

Кроме этого, планировщик следит, чтобы потребление выделенного «бюджета» не занимало много времени (например, процесс может читать секторы, далеко отстоящие друг от друга), в противном случае очередь «наказывается» путем снятия с обработки до исчерпания своего бюджета.

В листинге 4.34 показано управление классами и приоритетами планировщиков CFQ и BFQ, а в листинге 4.35 — пропорциональное распределение пропускной способности между конкурирующими процессами.

Листинг 4.34. I/O классы процессов планировщиков CFQ и BFQ

```
fitz@ubuntu:~$ ionice -p $$
```

```
none: prio 0
```

```
fitz@ubuntu:~$ ionice -c best-effort -n 5 -p $$
```

```
fitz@ubuntu:~$ ionice -p $$
```

```
best-effort: prio 5
```

```
fitz@ubuntu:~$ sudo ionice -c realtime -n 3 -p $$
```

```
fitz@ubuntu:~$ ionice -p $$
```

```
realtime: prio 3
```

В листинге 4.35 проведены два опыта, в которых сравнивается распределение пропускной способности при чтений с дискового накопителя /dev/sdc между двумя одинаковыми процессами.

Если процессы имеют одинаковый класс и приоритет, то в результате и получают одинаковую долю пропускной способности диска , а если разные приоритеты, то пропорциональную.

В принципе доля пропускной способности должна была распределиться как $40 \text{ мс}/(40 \text{ мс} + 180 \text{ мс}) = 0,18$ и $180 \text{ мс}/(40 \text{ мс} + 180 \text{ мс}) = 0,82$ согласно длительностям интервалов обработки, выделяемой очередям планировщиком (см. выше).

Полученный результат отличается от прогнозируемого потому, что один процесс заканчивает копирование раньше другого на целых 4 с, что составляет примерно 30% общего времени копирования, поэтому второй заканчивает свое чтение, уже используя полную пропускную способность диска.

Поэтому средние (!) скорости копирования в 1,5 и 2,2 Мбит/с и не соотносятся как 0,18 к 0,82.

Листинг 4.35. Распределение пропускной способности планировщиками CFQ и BFQ

```
fitz@ubuntu:~$ findmnt -T .
TARGET SOURCE      FSTYPE OPTIONS
/      /dev/sda4  ext4  rw,relatime,errors=remount-ro
fitz@ubuntu:~$ cat /sys/block/sda/queue/scheduler
noop deadline [cfq]
fitz@ubuntu:~$ dd if=/dev/urandom of=big1 > bs=16384 count=1024
1024+0 записей получено
1024+0 записей отправлено
16777216 bytes (17 MB, 16 MiB) copied, 0,089367 s, 188 MB/s
fitz@ubuntu:~$ dd if=/dev/urandom of=big2 > bs=16384 count=1024
...          ...
...          ...
fitz@ubuntu:~$ sync
fitz@ubuntu:~$ dd if=big1 of=/dev/null iflag=direct &
fitz@ubuntu:~$ dd if=big2 of=/dev/null iflag=direct &
```

Листинг 4.35. Продолжение.

```
fitz@ubuntu:~$ wait
[1] 6452
[2] 6453
32768+0 записей получено
32768+0 записей отправлено
16777216 bytes (17 MB, 16 MiB) copied, 12,6594 s, 1,3 MB/s
32768+0 записей получено
32768+0 записей отправлено
16777216 bytes (17 MB, 16 MiB) copied, 13,366 s, 1,3 MB/s
[1]- Завершён      dd if=big1 of=/dev/null iflag=direct
[2]+ Завершён      dd if=big2 of=/dev/null iflag=direct

fitz@ubuntu:~$ ionice -c best-effort -n 7 dd if=big1 of=/dev/null iflag=direct &
fitz@ubuntu:~$ ionice -c best-effort -n 0 dd if=big2 of=/dev/null iflag=direct &
fitz@ubuntu:~$ wait
...
...
...
16777216 bytes (17 MB, 16 MiB) copied, 7,45967 s, 2,2 MB/s
...
...
...
16777216 bytes (17 MB, 16 MiB) copied, 11,4372 s, 1,5 MB/s
...
...
...
```

Память процесса

Еще одним ресурсом, подлежащим распределению между процессами, является оперативная память.

В Linux, как и во многих других современных операционных системах, для управления памятью используют механизм страничного отображения, реализуемого ядром операционной системы при помощи устройства управления памятью — W:[MMU].

При этом процессы работают с виртуальными адресами (virtual address) «воображаемой» памяти, отображаемыми устройством MMU на физические адреса (physical address) настоящей оперативной памяти.

Для отображения (рис. 4.3) вся оперативная память (RAM) условно разбивается на «гранулы» — страничные кадры размером 4 Кбайт, которые затем выделяются процессам.

Таким образом, память процесса условно состоит из страниц (page), которым в специальных таблицах страниц (page table) сопоставлены выделенные страничные кадры (page frame).

При выполнении процесса преобразование его виртуальных адресов в физические выполняется устройством MMU «на лету» при помощи его индивидуальной таблицы страниц.

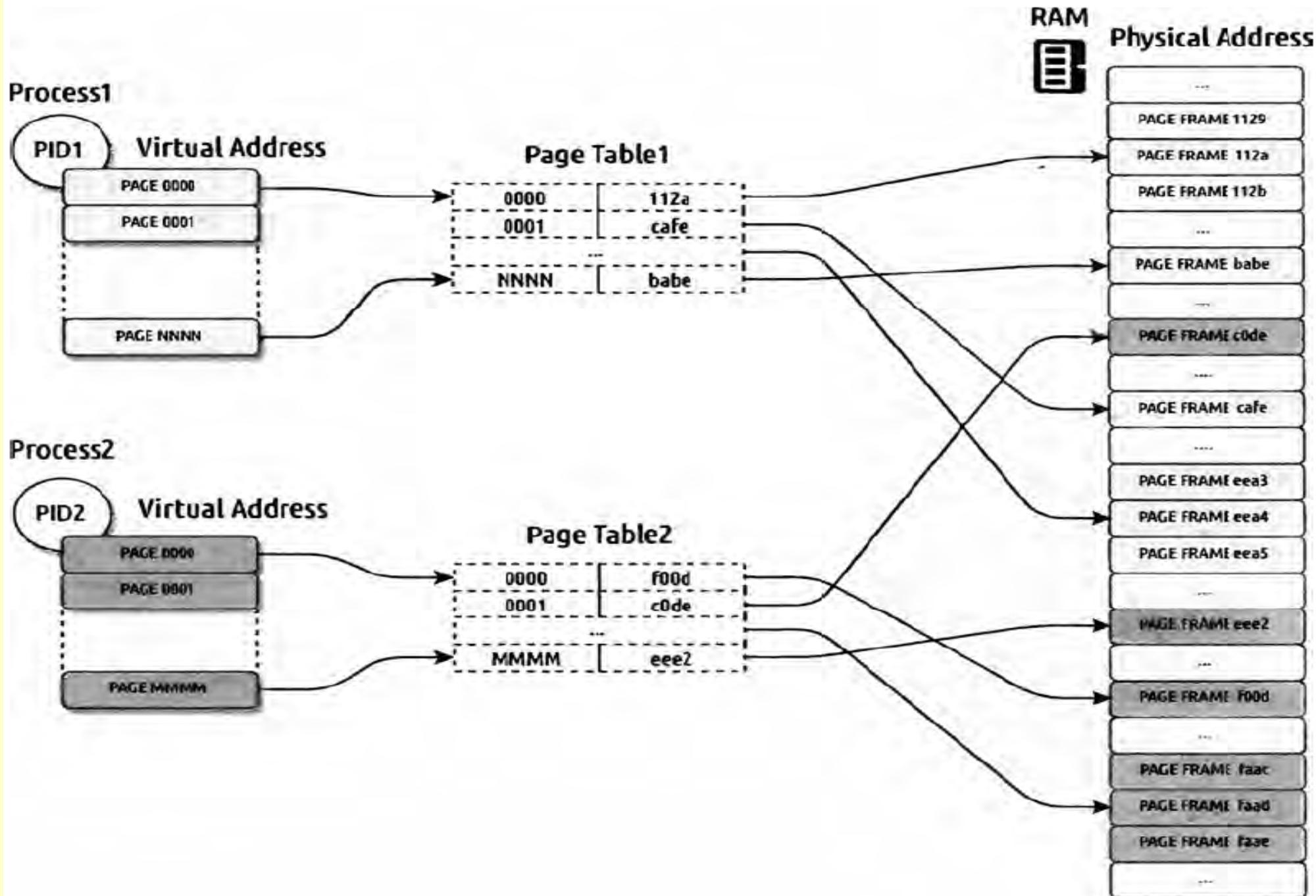


Рис. 4.3. Страницное отображение и распределение памяти

Именно механизм страничного отображения позволяет эффективно распределять память между процессами путем размещения страниц процессов в произвольные свободные страничные кадры.

Кроме этого, механизм страничного отображения позволяет выделять процессам память по требованию, добавляя дополнительные страницы и отображая их на свободные страничные кадры.

Аналогично, ненужные процессу страницы могут быть удалены, а соответствующие страничные кадры высвобождены для использования другими процессами.

Виртуальная память

Помимо задачи распределения памяти между процессами, механизм страничного отображения используется ядром операционной системы и для решения задачи нехватки оперативной памяти.

При определенных обстоятельствах имеющиеся в распоряжении свободные страничные кадры оперативной памяти могут быть исчерпаны.

Одновременно с этим оказывается, что большую часть времени процессы используют лишь малую часть выделенной им памяти, а находясь в состоянии сна, не используют память вовсе.

Увеличить коэффициент полезного использования памяти позволяет еще одна простая идея (рис. 4.4) — высвобождать страничные кадры при помощи выгрузки (page out) неиспользуемых страниц процессов во вторичную память (в специальную область «подкачки» SWAP, например, на диске), а при обращении к выгруженной странице — загружать (page in) ее обратно перед использованием.

За счет такого страничного обмена (paging или page swapping) организуется W: [виртуальная память], т. е. видимость большего количества (оперативной) памяти для размещения процессов, чем есть на самом деле.

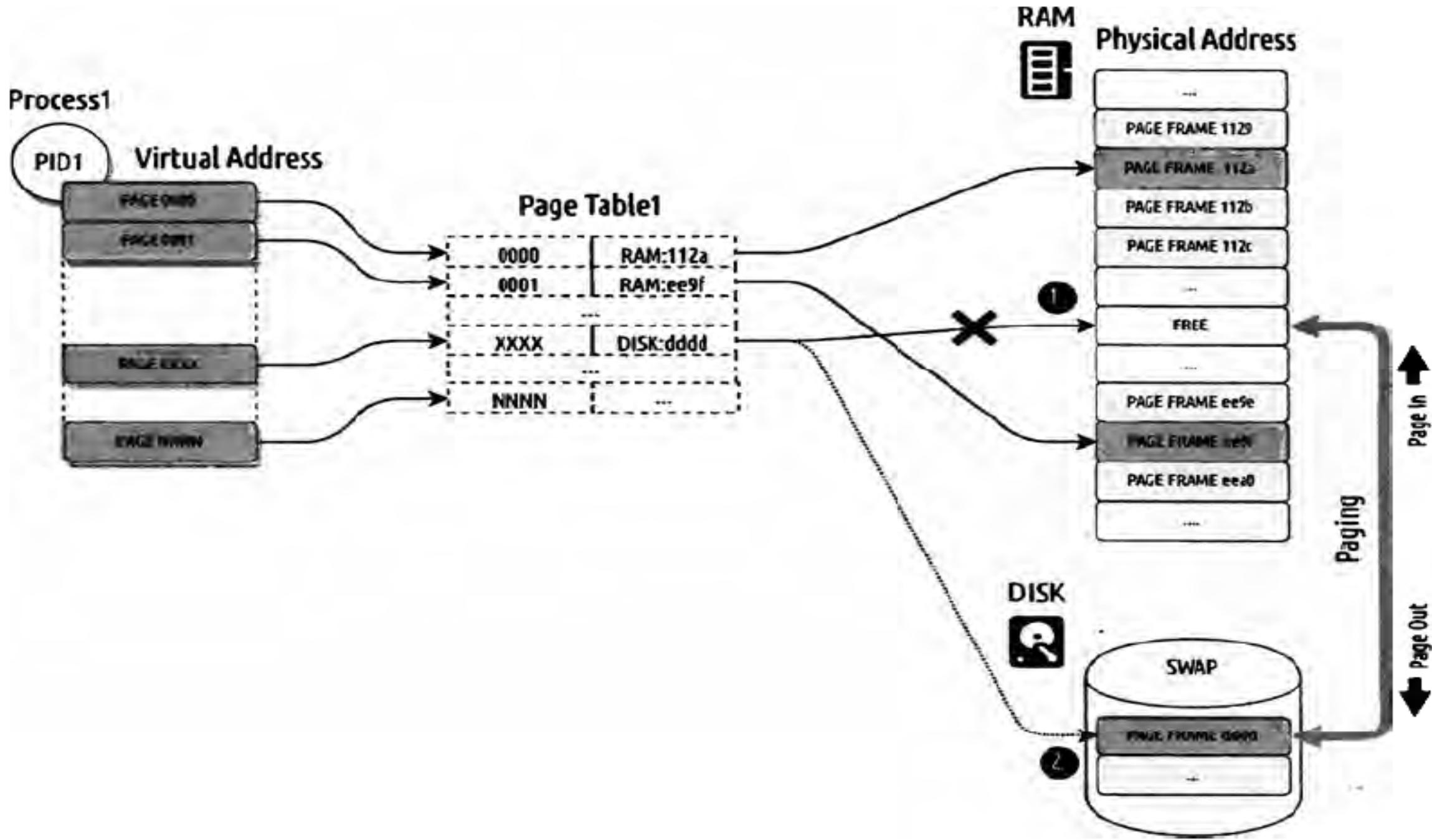


Рис. 4.4. Страницочный обмен

В примере из листинга 4.36 столбцах VSZ и RSS вывода команды ps показано потребление памяти процессами (в килобайтах).

В столбце VSZ (virtual size) указывается суммарный объем всех страниц процесса (в том числе и выгруженных), а в столбце RSS (resident set size) — суммарный объем всех его страницных кадров в оперативной памяти, т. е. ее реальное потребление процессом.

Листинг 4.36.

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
jake	4257	1.4	1.2	962016	49072	?	Ssl	20:46	0:00	\ .../gnome-terminal-server
jake	4267	0.0	0.1	12948	4808	pts/0	Ss+	20:46	0:00	\ bash
jake	4276	20.5	5.9	2930660	240788	?	Sl	20:46	0:05	\ .../firefox -new-window
jake	4378	8.6	3.5	2427016	141300	?	Sl	20:46	0:01	\ .../firefox ...
jake	4475	6.8	2.6	2390580	107920	?	Sl	20:46	0:01	\ .../firefox ...
jake	4521	3.2	2.0	2374856	84480	?	Sl	20:46	0:00	\ .../firefox ...

Отображение файлов в память

Страницочный обмен, помимо организации виртуальной памяти, имеет еще одно важнейшее применение.

Именно на его основе реализуется незаменимый механизм отображения файлов в память процесса, доступный при помощи системных вызовов mmap/munmap (и дополнительных mlock, mprotect, msync, madvise и др.).

Для отображения файла (рис. 4.5) в память процесса ему выделяют страницы в необходимом количестве (VSZ), но не страницные кадры (RSS).

Вместо этого в таблице страниц формируются такие записи, как будто эти страницы уже были выгружены (!) в отображаемый файл .

При последующем обращении (ondemand) процесса к какой-либо странице отображенной памяти под нее выделяют страницный кадр и заполняют (read) соответствующим содержимым файла.

Любые последующие изменения, сделанные процессом в отображенных страницах, сохраняются обратно (write back) в файл, если отображение выполнено «разделяемым» (shared) способом.

Для страниц, отображенных «частным» (private) способом, используется принцип COW (copy-on-write) , согласно которому любые попытки их изменения (write) приводят к созданию их копий (copy), куда и попадают изменения.

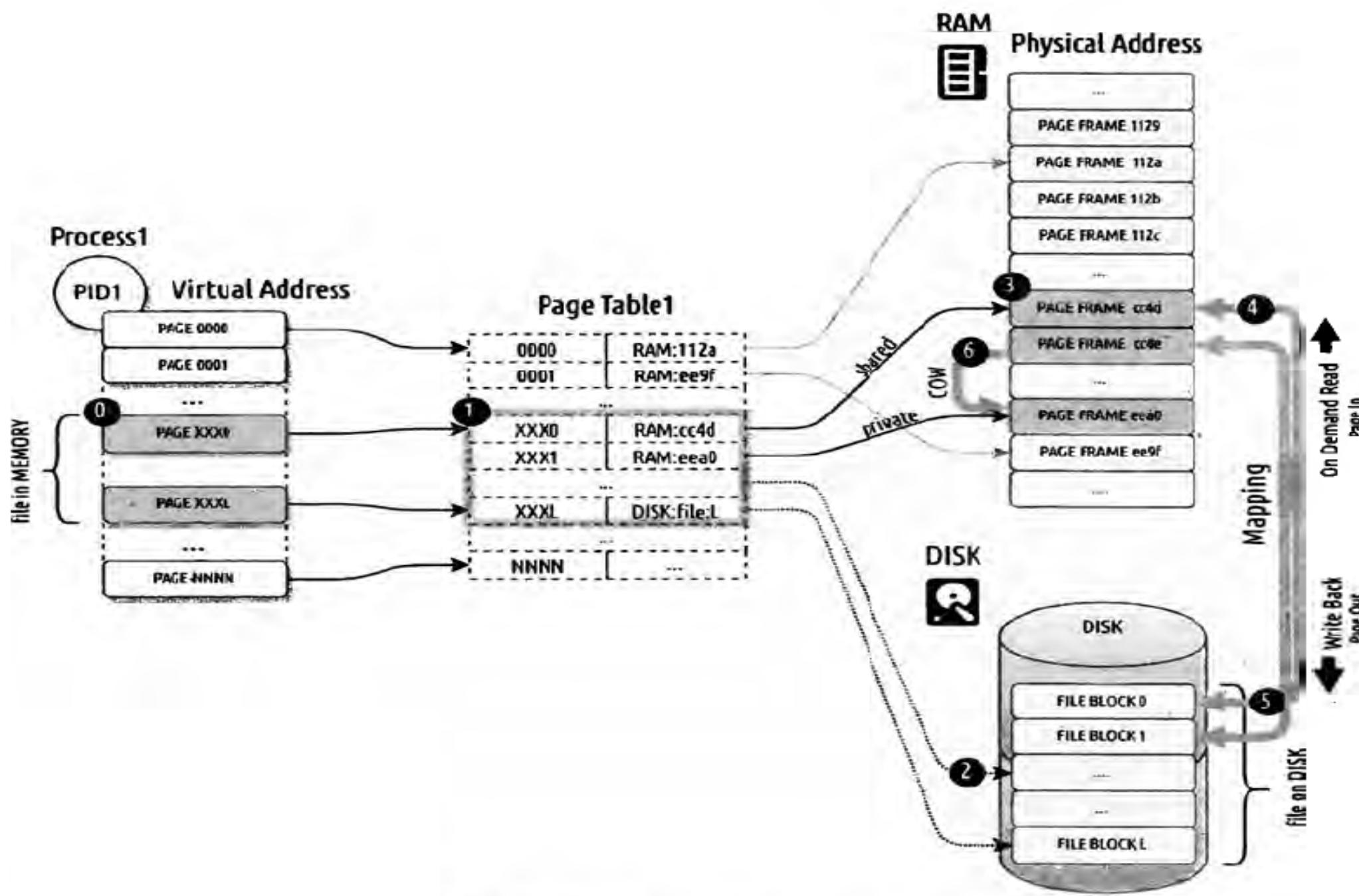


Рис. 4.5. Отображение файла в память

Таким образом, страницы отображенного файла, которые никогда не были востребованы процессом, не будут вовсе занимать оперативной памяти (не попадут в RSS).

Это обстоятельство широко используется для «загрузки» в процесс его программы и библиотек.

В листинге 4.37 при помощи команды `rtar` показана карта (отображения файлов) памяти процесса командного интерпретатора `bash`.

Листинг 437. Карта памяти процесса

```
fitz@ubuntu:~$ ps f
```

PID	TTY	STAT	TIME	COMMAND
26958	pts/0	Ss	0:00	bash
28540	pts/0	R+	0:00	_ ps f

```
fitz@ubuntu:~$ which bash
```

```
/usr/bin/bash
```

```
fitz@ubuntu:~$ readelf -l /usr/bin/bash
```

```
... ... ...
```

Заголовки программы:

Тип	Смеш.	Вирт.адр	Физ.адр		
	Рэм.файл	Рэм.пм	Флаги	Выравн	
LOAD	0x00000000002d000	0x00000000002d000	0x00000000002d000		
	0x0000000000ad78d	0x0000000000ad78d	R E	0x1000	
LOAD	0x0000000000110cf0	0x0000000000111cf0	0x0000000000111cf0		
	0x0000000000b914	0x0000000000155a8	RW	0x1000	
		

Листинг 437. Карта памяти процесса

```
fitz@ubuntu:~$ pmap -d 26958
```

```
26958: bash ↴
```

Адрес	Kб Mode	Offset	Device	Mapping
0000563b6b512000	180 r----	0000000000000000	000:00002	bash
0000563b6b53f000	696 r-x--	0000000002d000	000:00002	bash
0000563b6b5ed000	216 r----	000000000db000	000:00002	bash
0000563b6b623000	16 r----	00000000011000	000:00002	bash
0000563b6b627000	36 rw---	00000000011400	000:00002	bash
0000563b6b630000	40 rw---	0000000000000000	000:00000	[anon]
0000563b6b8d6000	1168 rw---	0000000000000000	000:00000	[anon]
	
00007f3b6267a000	148 r----	0000000000000000	000:00002	libc-2.30.so
00007f3b6269f000	1504 r-x--	0000000000025000	000:00002	libc-2.30.so
00007f3b62817000	296 r----	000000000019d000	000:00002	libc-2.30.so
00007f3b62861000	12 r----	00000000001e6000	000:00002	libc-2.30.so
00007f3b62864000	12 rw---	00000000001e9000	000:00002	libc-2.30.so
	
00007fff05018000	132 rw---	0000000000000000	000:00000	[stack]
mapped: 12932K	writeable/private: 1468K	shared: 28K		

В память процесса интерпретатора отображен исполняемый ELF-файл его программы и ELF-файлы всех библиотек , от которых она зависит.

Отображение ELF-файлов выполняется частями — сегментами (при помощи readelf можно получить их список), в зависимости от их назначения.

Так, например, сегмент программного кода отображен в страницы, доступные на чтение r и выполнение x, сегмент данных отображен в страницы, доступные на чтение r и запись w, и т. д.

Более того, выделение страниц памяти по требованию (heap, куча) в процессе работы процесса реализуется при помощи «воображаемого» отображения некоторого несуществующего, «анонимного файла» [anon] на страничные кадры.

Необходимо отметить, что механизм виртуальной памяти при освобождении неиспользуемых страниц выгружает в специальную область подкачки SWAP (см. рис. 4.4) только «анонимные» страничные кадры и «анонимизированные», полученные копированием при изменении (согласно принципу COW).

Неанонимные измененные кадры выгружаются непосредственно в соответствующие им файлы, а неизмененные освобождаются вовсе без выгрузки, т. к. уже «заранее выгружены».

В примере из листинга 4.38 иллюстрируются два способа выделения памяти по требованию: явный — при помощи системного вызова `mmap(2)` с аргументом `MAP_ANONYMOUS`, и неявный — при помощи системного вызова `brk`.

Явный способ позволяет выделять новые сегменты памяти процесса, тогда как неявный способ изменяет размер предопределенного «сегмента данных» процесса, позволяя увеличивать и уменьшать его по желанию, перемещая так называемый «`break`» — адрес конца этого сегмента.

Листинг 4.38. Системные вызовы mmap/munmap и brk - выделение и высвобождение памяти

```
fitz@ubuntu:~ $ ldd /usr/bin/hostname
    linux-vdso.so.1 (0x00007ffca19a6000)
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f5cd5da6000)
    /lib64/ld-linux-x86-64.so.2 (0x00007f5cd5fb3000)

fitz@ubuntu:~$ strace hostname
execve("/usr/bin/hostname", ["hostname"], 0x7ffc63e59df0 /* 31 vars */) = 0
brk(NULL)                                = 0x55fb3dc7b000
...                                         ...
...                                         ...
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=71063, ...}) = 0
mmap(NULL, 71063, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f48ae3b7000 ①
close(3)                                    = 0
...                                         ...
...                                         ...
...                                         ...
```

Листинг 4.38. Продолжение

```
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x1), ...}) = 0
write(1, "ubuntu\n", 7)          = 7
ubuntu
exit_group(0)                  = ?
+++ exited with 0 +++
```

Трасса команды `hostrrame()`, показанная в листинге 4.38, поясняет работу загрузчика и компоновщика (`loader`, `ld`) динамических библиотек `ld-linux`.

Системный вызов `exec` отображает для запуска в память процесса не только заданный ELF-файл `/bin/hostname`, но и (указанный в этом ELF-файле) загрузчик библиотек `/lib64/ld-linux-x86-64.so.2`, которому и передаются управление до самой программы.

Загрузчик библиотек, в свою очередь, отображает в процесс свой «конфигурационный» файл `/etc/ld.so.cache`, а затем посегментно отображает файлы всех библиотек и выделяет им требуемую дополнительную память.

Загруженные библиотеки присоединяются (линуются или же компонуются, `linking`) к программе `/bin/hostname`, после чего страницам их отображенных сегментов назначается соответствующий режим доступа системным вызовом `mprotect`.

По завершении компоновки отображение конфигурационного файла `/etc/ld.so.cache` снимается при помощи тиртма, а управление передается исходной программе.

Потребление памяти

Суммарное распределение страниц памяти по сегментам процесса можно получить при помощи третьего набора столбцов (активировав его специальной комбинацией клавиш, а затем добавив столбец SWAP клавишей F) команды top, как показано в листинге 4.39.

В столбце VIRT (VSZ в терминах ps) изображается суммарный объем (в килобайтах) всех страниц процесса, а в столбце RES (RSS в терминах ps) — объем резидентных страниц (находящихся в страничных кадрах оперативной памяти).

В столбце SWAP указывается объем всех страниц, находящихся во вторичной памяти — как «анонимных» страниц, выгруженных в специальную область подкачки, так и «файловых» страниц, возможно никогда не загружавшихся в оперативную память.

Столбцы CODE и DATA показывают объемы (в килобайтах) памяти, выделенных под сегменты кода и данных, а столбец SHR — объем резидентных страниц, которые используется (или могут быть использованы) совместно с другими процессами.

Листинг 4.39. Распределение памяти по назначению

```
fitz@ubuntu:~$ top -p 26958
```

```
[top - 22:10:12 up 1:48, 2 users, load average: 0,01, 0,02, 0,00
```

```
Tasks: 1 total, 0 running, 1 sleeping, 0 stopped, 0 zombie
```

```
%Cpu(s): 0,0 us, 0,0 sy, 0,0 ni,100,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
```

```
MiБ Mem : 3935,6 total, 1356,2 free, 974,9 used, 1604,4 buff/cache
```

```
MiБ Swap: 448,5 total, 448,5 free, 0,0 used. 2679,9 avail Mem
```

PID	%MEM	VIRT	SWAP	RES	CODE	DATA	SHR	nMaj	nDRT	%CPU	COMMAND
7019	0,1	13060	0	5108	876	1600	3600	0	0	0,0	bash

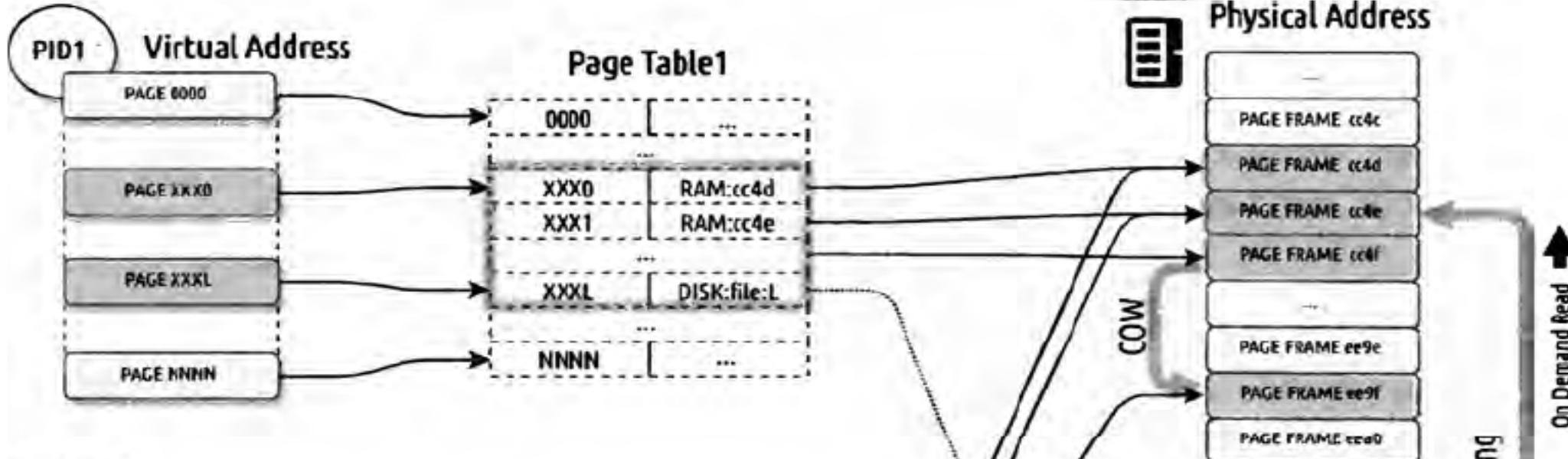
Механизм отображения считывает содержимое файла в страничные кадры только один раз, вне зависимости от количества процессов, отображающих этот файл в свою память.

В случае отображения одного файла разными процессами (рис. 4.6) их страницы совместно отображаются на одни и те же страничные кадры, за исключением страниц, скопированных (согласно принципу COW) при изменении.

Такое поведение механизма отображения позволяет эффективно расходовать оперативную память за счет использования разными программами одинаковых разделяемых библиотек.

Так как ELF-файлы библиотек «загружаются» в память процессов при помощи отображения mmap, то в результате каждая библиотека размещается в оперативной памяти лишь единожды, вне зависимости от количества ее «использований».

Process1



Process2

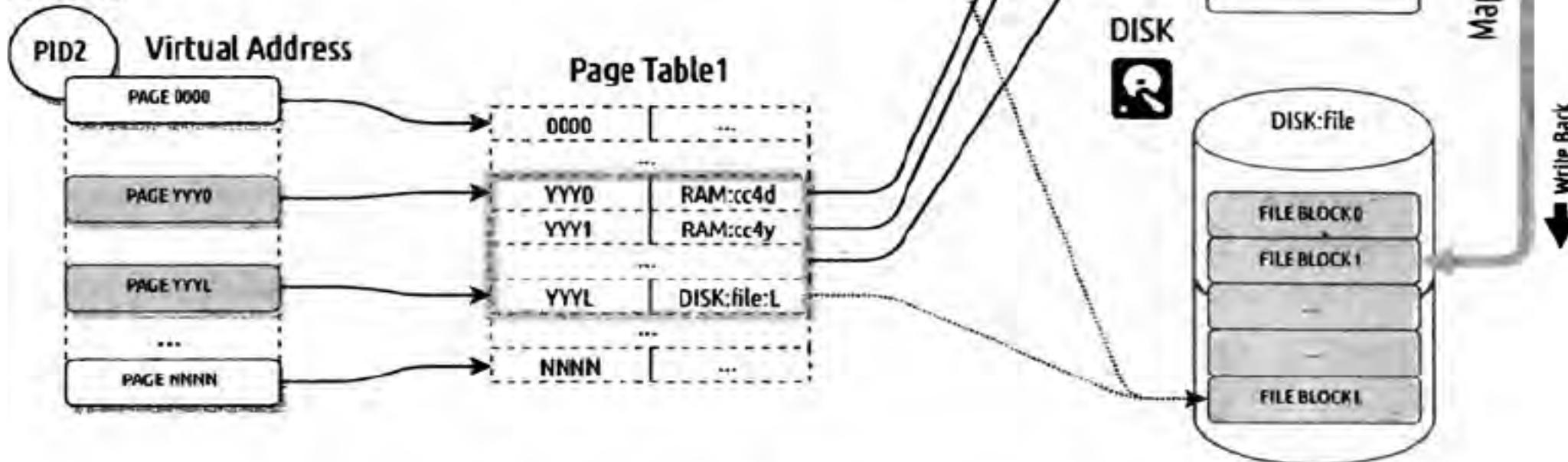


Рис. 4.6. Совместное использование памяти

Интегральная статистика по использованию виртуальной памяти может быть получена при помощи команды free, как показано в листинге 4.40.

Листинг 4.40. Статистика использования память

```
fitz@ubuntu:~$ free -m
```

	всего	занято	свободно	общая	буферы	временные	доступно
Память:	3935	975	1354	39	71	1534	2679
Подкачка:	448	0	448				

```
fitz@ubuntu:~$ LANGUAGE=en free -m
```

	total	used	free	shared	buffers	cache	available
Mem:	3935	975	1354	39	71	1534	2679
Swap:	448	0					

Строка Mem:- содержит статистику использования оперативной памяти, а строка Swap: — статистику специальной области подкачки.

В столбце total указан суммарный объем всех доступных страницных кадров, а в столбцах used (вычисляется как used = total -free-buffers- cache) и free — суммарные объемы использованных и свободных страницных кадров соответственно.

В столбце cache указан объем страницного кэша (page cache), т. е. суммарный объем страницных кадров оперативной памяти, использованных под отображение файлов в память.

Необходимо заметить, что именно страницный кэш является неким резервом памяти, которую можно высвободить при первой необходимости, т. к. содержимое этих страниц в реальности всегда можно считать из отображаемых файлов заново (и то, если понадобится).

Аналогично, в столбце buffers указывается объем буферного кэша, т. е. суммарный объем памяти, использованной ядром для кэширования «неотображаемых» сущностей: метаданных файлов, дисковых блоков при прямом вводе-выводе на устройства и пр.

В столбце `available` предсказывается объем доступной памяти для новых процессов, без вытеснения страниц старых.

В листинге 4.41 показан пример потребления памяти процессом текстового редактора `vi` при попытке редактирования громадного файла в 1 Гбайт. Процесс загружает файл целиком, в результате чего он целиком оказывается в резидентных страницах сегмента данных процесса, что естественным образом увеличивает «чистый» расход оперативной памяти системы. После принудительного завершения процесса при помощи команды `kill` память естественным образом высвобождается.

Однако, так как под процесс редактора был высвобожден страничный кэш, повторное его наполнение будет происходить позже, при обращении процессов к своим отображенными файлам (и то, если потребуется).

Листинг 4.41. Потребление памяти процессами

```
fitz@ubuntu:~$ dd if=/dev/urandom of=big bs=4069 count=262144
262144+0 записей получено
262144+0 записей отправлено
1066663936 байт (1,1 GB, 1017 MiB) скопирован, 13,3567 s, 79,9 MB/s
```

```
fitz@ubuntu:~$ ls -lh big
```

```
-rw-rw-r-- 1 fitz fitz 1018M ноя 19 23:03 big
```

```
fitz@ubuntu:~$ free -m
```

	total	used	free	shared	buffers	cache	available
Mem:	3935	987	509	39	71	2366	2656
Swap:	448	0	448				

```
fitz@ubuntu:~$ vi big
```

```
fitz@ubuntu:~$ ps f
```

PID	TTY	STAT	TIME	COMMAND
20595	pts/1	S	0:00	-bash
21087	pts/1	R+	0:00	__ ps f
20437	pts/0	S	0:00	-bash
21085	pts/0	Rl+	0:08	__ vi big

Листинг 4.41 Продолжение

```
fitz@ubuntu:~$ ps up 21085
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
fitz	21085	96.4	21.8	1816164	1807248	pts/0	Sl+	21:14	0:18	vi big

```
fitz@ubuntu:~$ free -m
```

	total	used	free	shared	buffers	cache	available
Mem:	3935	2752	133	38	55	993	913
Swap:	448	6	442				

```
fitz@ubuntu:~$ top -b -n1 -p 21085
```

```
top - 23:11:05 up 2:49, 3 users, load average: 0,00, 0,09, 0,07
```

```
Tasks: 1 total, 0 running, 1 sleeping, 0 stopped, 0 zombie
```

```
%Cpu(s): 0,0 us, 0,0 sy, 0,0 ni, 100,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
```

Листинг 4.41 Продолжение

MiB Mem : 3935,6 total, 133,0 free, 2753,0 used, 1049,6 buff/cache

MiB Swap: 448,5 total, 442,0 free, 6,5 used. 913,0 avail Mem

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+ COMMAND
7680	fitz	20	0	1816164	1.7g	3424	S	0,0	44,8	0:14.90 vi

fitz@ubuntu:~\$ kill 21085

fitz@ubuntu:~\$ free -m

	total	used	free	shared	buffers	cache	available
Mem:	3935	986	1908	38	56	984	2679
Swap:	448	6	442				

Механизм сигналов

Механизм сигналов `signal` является простейшей формой межпроцессного взаимодействия и предназначен для внешнего управления процессами.

Каждый сигнал имеет свой обработчик, определяющий поведение процесса при отсылке ему этого сигнала.

Этот обработчик является неким набором инструкций программы (подпрограммой), которым передается управление при доставке сигнала процессу.

Каждому процессу назначаются обработчики «по умолчанию», в большинстве случаев приводящие к завершению процесса.

В примере из листинга 4.42 показано применение сигнала штатного прерывания процесса № 2 (`SIGINT`), отсылаемого драйвером терминала всем процессам «переднего» фона при получении символа `^C(ETX)`, т. е. нажатии клавиш `Ctrl+C` на терминале.

Для процессов «заднего» фона сигнал может быть отослан явно, при помощи команды `kill`, предназначеннай, несмотря на ее название, для отсылки сигналов.

Листинг 4.42. Штатное прерывание процесса (^C, intr, SIGINT)

```
fitz@ubuntu:~$ stty -a
speed 38400 baud; rows 24; columns 80; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>; ...
...           ...           ...
isig ~icanon iexten echo echoe echok -echonl -noflsh -xcase -tostop ... -extproc
```

```
fitz@ubuntu:~$ dd if=/dev/zero of=/dev/null
^C782349+0 записей получено
```

Листинг 4.42. Продолжение

```
782349+0 записей отправлено  
скопировано 400562688 байт (401 MB), 0,531532 c, 754 MB/c
```

```
fitz@ubuntu:~$ dd if=/dev/zero of=/dev/null &  
[1] 23418  
fitz@ubuntu:~$ ^C  
...  
fitz@ubuntu:~$ ^C  
...  
fitz@ubuntu:~$ jobs -l  
[1]+ 23418 Запущен dd if=/dev/zero of=/dev/null &  
  
fitz@ubuntu:~$ kill -SIGINT 23418  
fitz@ubuntu:~$ 25318811+0 записей получено ^C  
25318810+0 записей отправлено  
скопировано 12963230720 байт (13 GB), 17,1001 c, 758 MB/c  
  
[1]+ Прерывание dd if=/dev/zero of=/dev/null
```

Нужно отметить, что настройки драйвера терминала позволяют как переопределить символ, получение которого приведет к выполнению действия `intr` (посылка сигнала `SIGINT`), так и совсем выключить отсылку подобных сигналов управления процессами при получении управляющих символов.

В листинге 4.43 показано аналогичное применение сигнала аварийного завершения процесса №3 (`SIGQUIT`), посылаемого процессам «переднего» фона при получении драйвером управляющего символа `^\\` (`FS`), генерируемого терминалом при нажатии `Ctrl+\\`.

Обработчик сигнала не только завершит процесс, но и попытается (если позволяют настройки) сохранить дамп памяти процесса в файл `coge` для последующей отладки.

Листинг 4.43. Аварийное прерывание процесса (^\\, quit,SIGQUIT)

```
fitz@ubuntu:~$ dd if=/dev/zero of=/dev/null  
^\\Выход (стек памяти сброшен на диск)  
fitz@ubuntu:~$ dd if=/dev/zero of=/dev/null &  
[1] 23429  
fitz@ubuntu:~$ kill -SIGQUIT 23429  
[1]+  Выход                  (стек памяти сброшен на диск)  dd if=/dev/zero of=/dev/null
```

Некоторые процессы (например, демоны, или графические приложения) не имеют управляющего терминала, поэтому не могут быть завершены интерактивно при помощи Ctrl+C или Ctrl+\.

В этом случае используется сигнал штатного завершения № 15 (SIGTERM), что проиллюстрировано в листинге 4.44.

Листинг 4.44. Штатное завершение процесса (SIGTERM)

```
fitz@ubuntu:~$ dd if=/dev/zero of=/dev/null &
[1] 23444
fitz@ubuntu:~$ kill -SIGTERM 23444
fitz@ubuntu:~$
[1]+  Завершено          dd if=/dev/zero of=/dev/null
```

Для приостановки процесса, т. е. временного исключения его из процедур распределения процессорного времени планировщиком, предназначен сигнал № 19 (SIGSTOP), а для возобновления процесса — сигнал № 18 (SIGCONT).

В листинге 4.45 показано, что приостановленный (sTopped) процесс не потребляет процессорного времени.

Листинг 4.45. Приостановка и возобновление процесса (SIGSTOP и SIGCONT)

```
fitz@ubuntu:~$ pbzip2 big &
[1] 1647
fitz@ubuntu:~$ top -b -n1 -p 1647
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 1647 fitz      20   0  102m  33m  900 S 387,0  0,4    0:25.09 pbzip2

fitz@ubuntu:~$ pkill -STOP pbzip2
fitz@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
 1640 pts/2    Ss      0:00  -bash
 1647 pts/2    TL    0:24  \_ pbzip2 big
 1651 pts/2    R+      0:00  \_ ps f
fitz@ubuntu:~$ jobs -l
[1]+  1647 Остановлен          pbzip2 big
fitz@ubuntu:~$ top -b -n1 -p 1647
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 1647 fitz      20   0  102m  34m  900 T  0,0  0,4    0:42.90 pbzip2

fitz@ubuntu:~$ kill -CONT 1647
fitz@ubuntu:~$ top -b -n1 -p 1647
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 1647 fitz      20   0  102m  34m  900 S 370,0  0,4    0:51.82 pbzip2
```

Сигналы могут быть перехвачены, если процесс назначит собственный обработчик, а также проигнорированы, тогда при их доставке вообще никакой обработчик не вызывается.

Иключение составляют некоторые «безусловные» сигналы, такие как № 9 (SIGKILL) — безусловное завершение или №19 (SIGSTOP) — безусловная приостановка процесса.

Игнорирование и перехват сигналов показаны в листинге 4.46 на примере командного интерпретатора bash.

Ни попытки «интерактивного» завершения при помощи сигналов SIGINT и SIGQUIT, ни явная посылка сигналов SIGINT и SIGTERM не приводят к завершению командного интерпретатора.

К желаемому результату приводит только явная отсылка сигнала SIGKILL.

Листинг 4.46. Игнорирование и перехват сигналов

```
fitz@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
23025 pts/1    S      0:00 -bash
23771 pts/1    R+     0:00 \_ ps f

fitz@ubuntu:~$ bash
fitz@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
23025 pts/1    S      0:00 -bash
23636 pts/1    S      0:00 \_ bash
23692 pts/1    R+     0:00     \_ ps f

fitz@ubuntu:~$ ^C
fitz@ubuntu:~$ ^\_
fitz@ubuntu:~$ ps f
  PID TTY      STAT   TIME COMMAND
23025 pts/1    S      0:00 -bash
23636 pts/1    S      0:00 \_ bash
23692 pts/1    R+     0:00     \_ ps f
```

Листинг 4.46 Продолжение

```
fitz@ubuntu:~$ kill -SIGINT 23636
```

```
fitz@ubuntu:~$ ps f
```

PID	TTY	STAT	TIME	COMMAND
23025	pts/1	S	0:00	-bash
23636	pts/1	S	0:00	_ bash
23708	pts/1	R+	0:00	_ ps f

```
fitz@ubuntu:~$ kill -SIGTERM 23636
```

```
fitz@ubuntu:~$ ps f
```

PID	TTY	STAT	TIME	COMMAND
-----	-----	------	------	---------

23025	pts/1	S	0:00	-bash
23636	pts/1	S	0:00	_ bash
23708	pts/1	R+	0:00	_ ps f

```
fitz@ubuntu:~$ kill -SIGKILL 23636
```

Убито

```
fitz@ubuntu:~$ ps f
```

PID	TTY	STAT	TIME	COMMAND
23025	pts/1	S	0:00	-bash
23771	pts/1	R+	0:00	_ ps f

Диспозицию сигналов, т. е. информацию об игнорировании (IGNORED), перехвате (CAUGHT), временной блокировке (BLOCKED) или ожидании доставки (PENDING) сигналов процессов можно получить при помощи команды ps, как показано в листинге 4.47.

Диспозиция изображается битовой маской, где каждый N-й бит маски (нумерация от младших к старшим) соответствует сигналу N.

Например, командный интерпретатор игнорирует сигналы, представленные (шестнадцатеричной) маской 0038400416, что в двоичном представлении составляет 11100001000000000001002 и указывает на игнорируемые 3-й, 15-й, 20-й, 21-й и 22-й сигналы, т. е. SIGQUIT, SIGTERM, SIGTSTP, SIGTTIN и SIGTT0U.

Для перевода из шестнадцатеричного в двоичное представление использован стековый калькулятор dc, которому было велено из входной i (input) системы счисления по основанию 16 в выходную o (output) систему счисления по основанию 2 напечатать p (print) число 06384604, а для просмотра имен сигналов по их номерам использована встроенная команда kill.

Листинг 4.47 Диспозиция сигналов

```
fitz@ubuntu:~$ ps s
```

UID	PID	PENDING	BLOCKED	IGNORED	CAUGHT	STAT	TTY	TIME	COMMAND
1006	23825	00000000	00010000	00380004	4b813efb	Ss	pts/5	0:00	-bash
1006	23773	00000000	00000000	00000000	<f3d1fef9	R+	pts/5	0:00	ps s

```
fitz@ubuntu:~$ dc -e 16l2o00380004p
```

1110000000000000000000100
22210-----87654321

```
fitz@ubuntu:~$ kill -l
```

1) SIGHUP	2) SIGINT	3) SIGQUIT	4) SIGILL	5) SIGTRAP
	
11) SIGSEGV	12) SIGUSR2	13) SIGPIPE	14) SIGALRM	15) SIGTERM
16) SIGSTKFLT	17) SIGCHLD	18) SIGCONT	19) SIGSTOP	20) SIGTSTP
21) SIGTTIN	22) SIGTTOU	23) SIGURG	24) SIGXCPU	25) SIGXFSZ
	
63) SIGRTMAX-164)	SIGRTMAX			

Архитектура операционной системы Андроид

Структура ОС Android.

Уровни операционной системы.

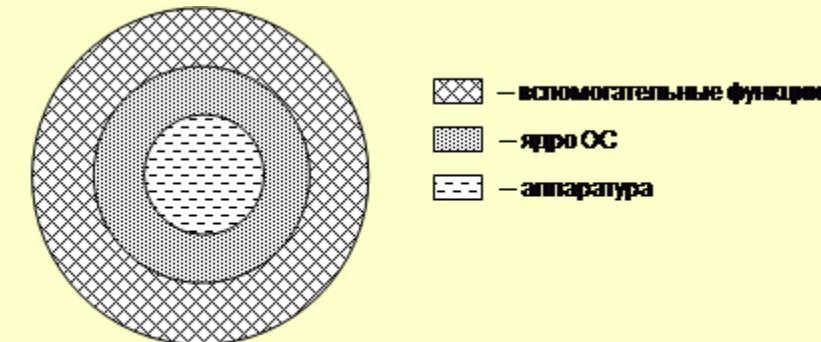
Решаемые задачи. Применения на различных
системах

Лекция

Многослойная структура ОС

Вычислительную систему, работающую под управлением ОС на основе ядра, можно рассматривать как систему, состоящую из трех иерархически расположенных слоев:

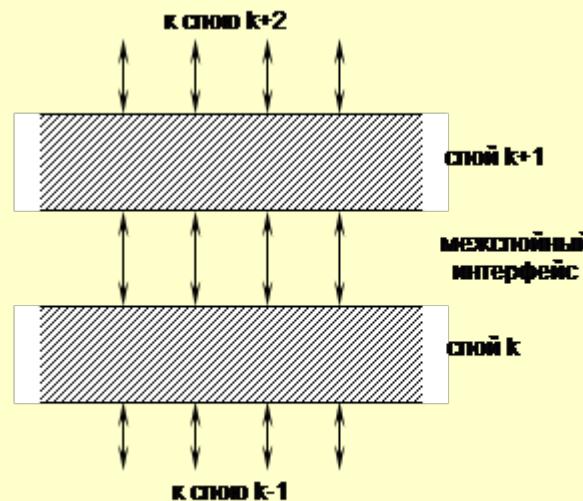
- нижний слой – образуется аппаратурой;
- промежуточный — ядро;
- верхний слой – модули, реализующие вспомогательные функции.



Трехслойная схема вычислительной системы

Слоистую структуру, изображенную на рисунке, удобно представлять для иллюстрации того факта, что каждый слой может взаимодействовать только с соседним слоем. При такой организации приложения не могут непосредственно взаимодействовать с аппаратурой, а только через слой ядра.

Многослойный подход является универсальным и эффективным способом декомпозиции сложных систем. В соответствии с ним система состоит из иерархии слоев. Каждый слой обслуживает вышележащий слой, выполняя для него некоторый набор функций, которые образуют межслойный интерфейс.



Концепция многослойного взаимодействия

На основе функций нижележащего слоя следующий слой строит свои функции — более сложные и более мощные, которые, в свою очередь, становятся примитивами для создания еще более мощных функций вышележащего слоя.

Строгие правила взаимодействия оговариваются только между слоям. Внутри слоя связи между модулями могут быть произвольными.

Отдельный модуль может выполнить свою работу как самостоятельно, как и обратиться к другому модулю своего слоя, либо к нижележащему слою через межслойный интерфейс.

Достоинства такой системы организации:

- упрощается разработка системы;
- при модернизации системы легко производить изменения внутри каждого слоя, не заботясь о других слоях.

Многослойная структура ядра ОС

Поскольку ядро представляет собой сложный многофункциональный комплекс, этот подход распространяют и на структуру ядра:

- **Средства аппаратной поддержки ОС.** Сюда относят не все аппаратные средства, а только те, которые прямо участвуют в организации вычислительных процессов: средства поддержки привилегированного режима, систему прерываний, средства переключения контекстов процессов, средства защиты областей памяти.
- **Машинно-зависимые компоненты ОС.** Слой образуют программные модули, в которых отражается специфика аппаратной платформы компьютера. В идеале этот слой полностью экранирует вышележащие слои ядра от особенностей аппаратуры. Это позволяет разрабатывать вышележащие слои на основе машинно-независимых модулей для всех типов аппаратных платформ, поддерживаемых данной ОС.
- **Базовые механизмы ядра.** Выполняет наиболее примитивные операции ядра, такие как переключение контекстов процессов, диспетчеризацию прерываний, перемещение страниц памяти на диск и обратно и т. п. Модули данного слоя не принимают решений о распределении ресурсов — исполнительные механизмы для модулей верхних слоев.

- **Менеджеры ресурсов.** Слой состоит из мощных функциональных модулей, реализующих стратегические задачи по управлению основными ресурсами вычислительной системы. Здесь работают менеджеры (диспетчеры) процессов, ввода-вывода, файловой системы и оперативной памяти. Разбиение на менеджеры может быть различным. например менеджер файловой системы иногда объединяют с менеджером ввода-вывода, а функции управления доступом пользователей к системе в целом и ее отдельным объектам поручают отдельному менеджеру безопасности.

Каждый из менеджеров ведет учет свободных и используемых ресурсов определенного типа и планирует их распределение в соответствии с запросами приложений.

Для исполнения принятых решений менеджер обращается к нижележащему слою. Внутри слоя менеджеров существуют тесные взаимосвязи, отражающие тот факт, что для выполнения процессу нужен доступ одновременно к нескольким ресурсам — например, процессору, памяти, вводу-выводу и т.п.

- **Интерфейс системных вызовов.** Это самый верхний слой ядра. Он взаимодействует непосредственно с приложениями и системными утилитами, образуя прикладной программный интерфейс ОС. Функции обслуживающие системные вызовы, предоставляют доступ к ресурсам системы в удобной и компактной форме, без указания деталей их физического расположения. Для осуществления таких действий системные вызовы обычно обращаются за помощью к функциям слоя менеджеров ресурсов и т.д.

Приведенное разбиение ядра ОС на слои – достаточно условно. Такое разбиение и распределение функций может быть иным.

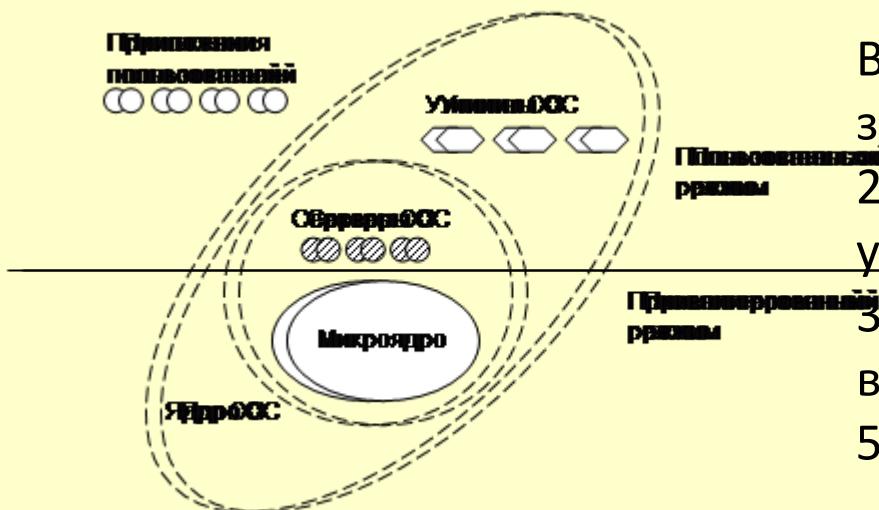
Также может быть иным способ взаимодействия слоев. Для ускорения работы ядра в ряде случаев происходит обращение с верхнего слоя к функциям слоев нижнего уровня, минуя промежуточные.

Выбор количества слоев ядра является ответственным и сложным делом: увеличение числа слоев ведет к замедлению работы ядра, а уменьшение – ухудшает расширяемость и логичность системы.

Микроядерная архитектура

Микроядерная архитектура – альтернатива классическому способу организации в виде многослойного ядра, работающего в привилегированном режиме.

Здесь в привилегированном режиме остается работать только очень небольшая часть ОС, называемая **микроядром**. Микроядро защищено от остальных частей ОС и приложений.



В состав микроядра входят 1.Машинно-зависимые модули; 2.Модули, выполняющие основные базовые функции ядра по управлению процессами; 3.Обработка прерываний; 4.Управлению виртуальной памятью; 5.Пересылка сообщений и управление вводом- выводом

Дизайн мобильных ОС прошел эволюцию от ОС для настольных рабочих станций через встраиваемые ОС до тех продуктов, которые мы видим в смартфонах сейчас.

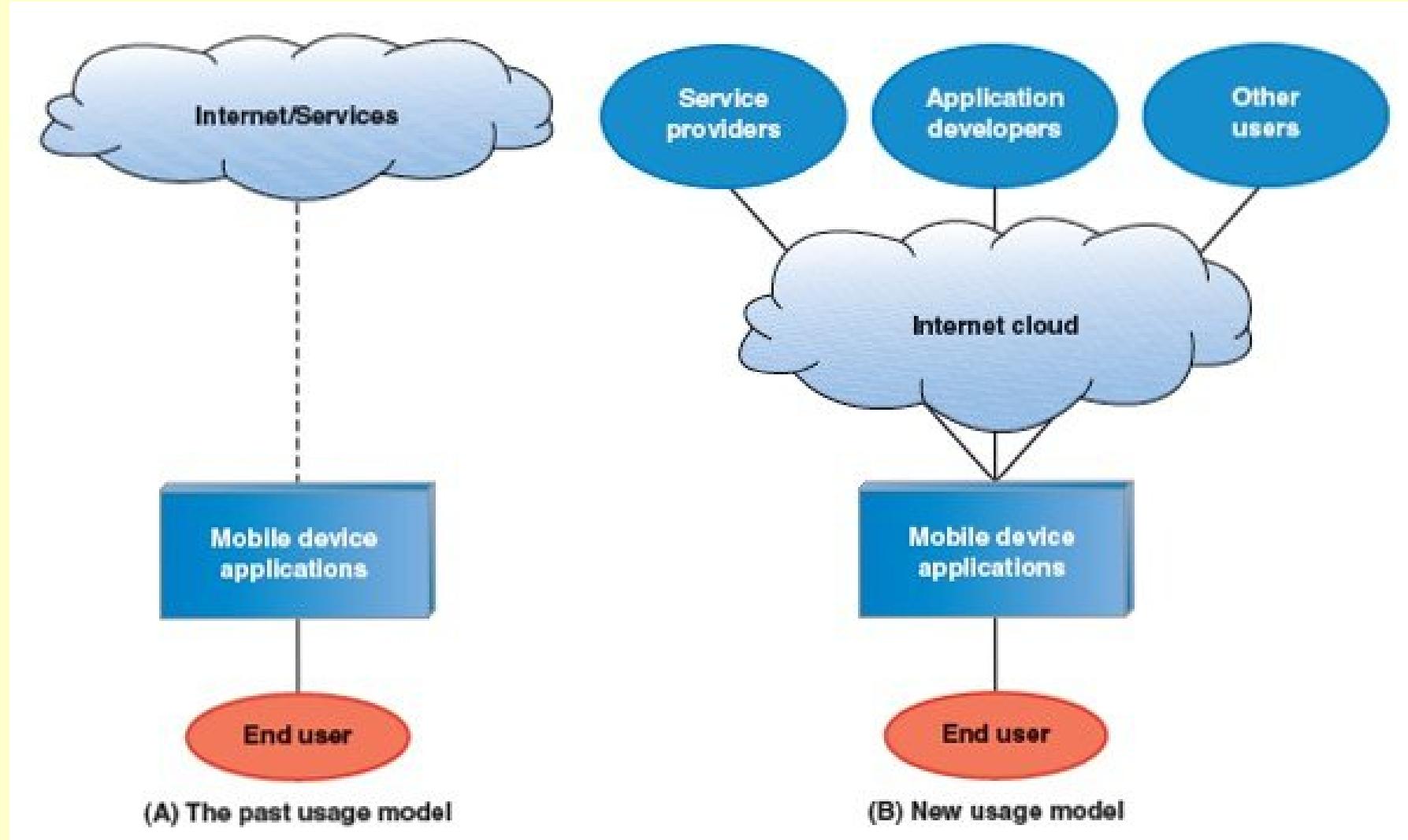
В течение этого процесса архитектура ОС менялась от сложной к простой и остановилась где-то на середине.

Сама же эволюция приводилась в движение технологическими достижениями в аппаратной и программной области, а также в интернет сервисах.

В недавнем прошлом модель использования мобильных устройств была весьма простой.

Пользователь запускал приложения для управления данными или оффлайновых игр, иногда загружал статические веб-странички или пользовался почтой.

Сейчас ситуация поменялась кардинальным образом: больше нет «предустановленных» функций, устройство выступает неким порталом в среду, где множество игроков – сервис провайдеры, независимые разработчики и т.д. – предоставляют огромное количество сервисов.



Модели использования мобильных устройств

С точки зрения моделей потребления, все представители мобильных ОС сегодняшнего дня (такие как Apple iOS, Google Android, Microsoft Windows) имеют больше сходных черт, нежели различий:

- Все они имеют документированные SDK с прописанными API, что позволяет разработчикам создавать приложения под эти ОС;
- Все они имеют он-лайн каталоги приложений, где разработчики публикуют свои приложения и откуда пользователи их скачивают;
- В каждой реализована многозадачность и поддержка 3D-графики, широко используются датчики и сенсорные экраны;
- Во всех системах большое внимание уделено гладкости и отзывчивости во взаимодействии с пользователем;
- Использование интернет далеко ушло от статических страниц, HTML5 становится платформой по умолчанию для Web-приложений;
- Все ОС поддерживают мобильные системы платежей;
- Все системы сфокусированы на оптимизации энергопотребления.

Общность нынешних мобильных ОС обусловлены глобальностью технологических трендов в аппаратной и программных областях, а также в коммуникациях.

Проанализируем теперь ОС нового поколения с точки зрения критериев, приведенных выше.

Отметим сразу, что в довольно большой степени они конфликтуют друг с другом.

Ощущения пользователей

Традиционное понятие производительности с трудом применимо к мобильным устройствам.

Вместо статической производительности по отношению к смартфонам логичнее оперировать понятием комфорtnости для пользователя, способностью оптимально реагировать на его действия, выраженной в чувствительности, плавности, логичности и точности работы.

Совершенно обычна ситуация, когда устройство А уступает Б по совокупности бенчмарков, однако с точки зрения пользовательского восприятия ценится выше, ведь тесты, измеряя те или иные подсистемы смартфона, не принимают в расчет взаимодействие с пользователем, а человек оценивает прежде всего это.

Возьмем для примера видео.

Традиционные тесты оперируют рядом метрик, таких как FPS или количество потерянных кадров.

В этом подходе есть как минимум две проблемы.

Первая: воспроизведение видео – это только одно действие из целого комплекса, включающего запуск плеера, загрузку в него видеоданных, процесс перемотки и т.д. С точки зрения пользователя, оценивать нужно все вместе.

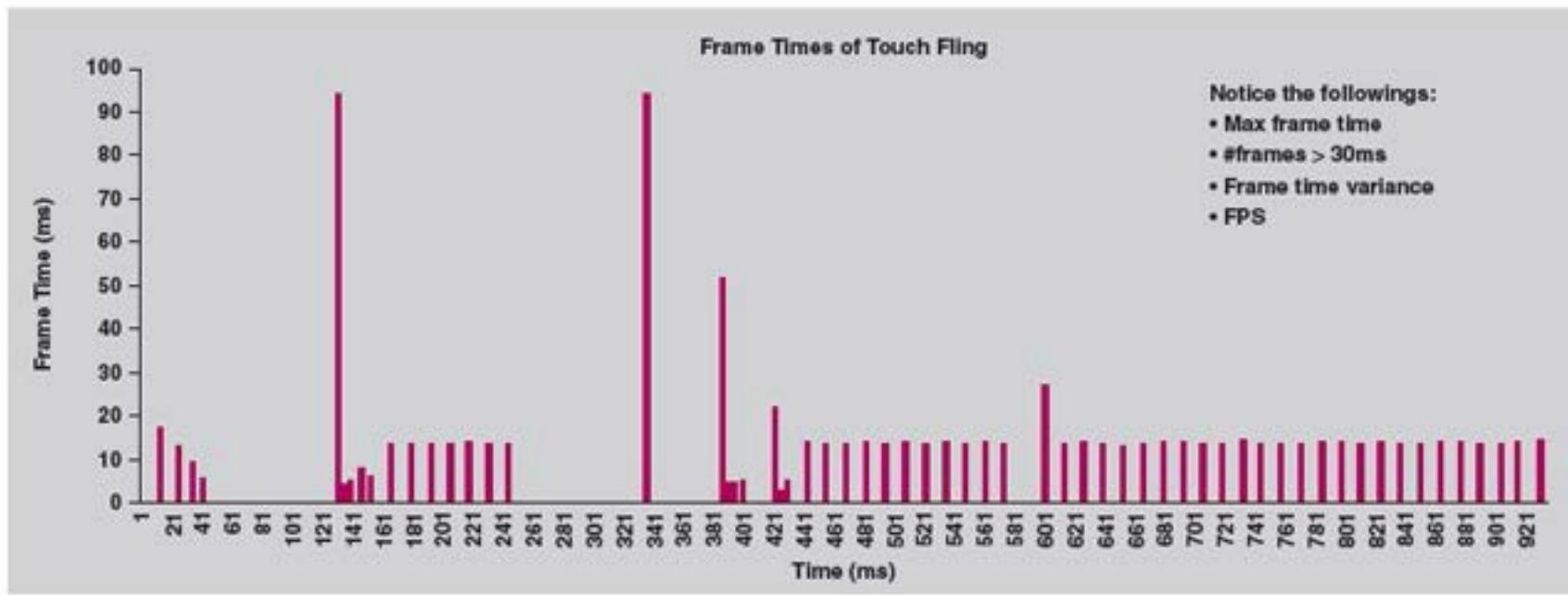
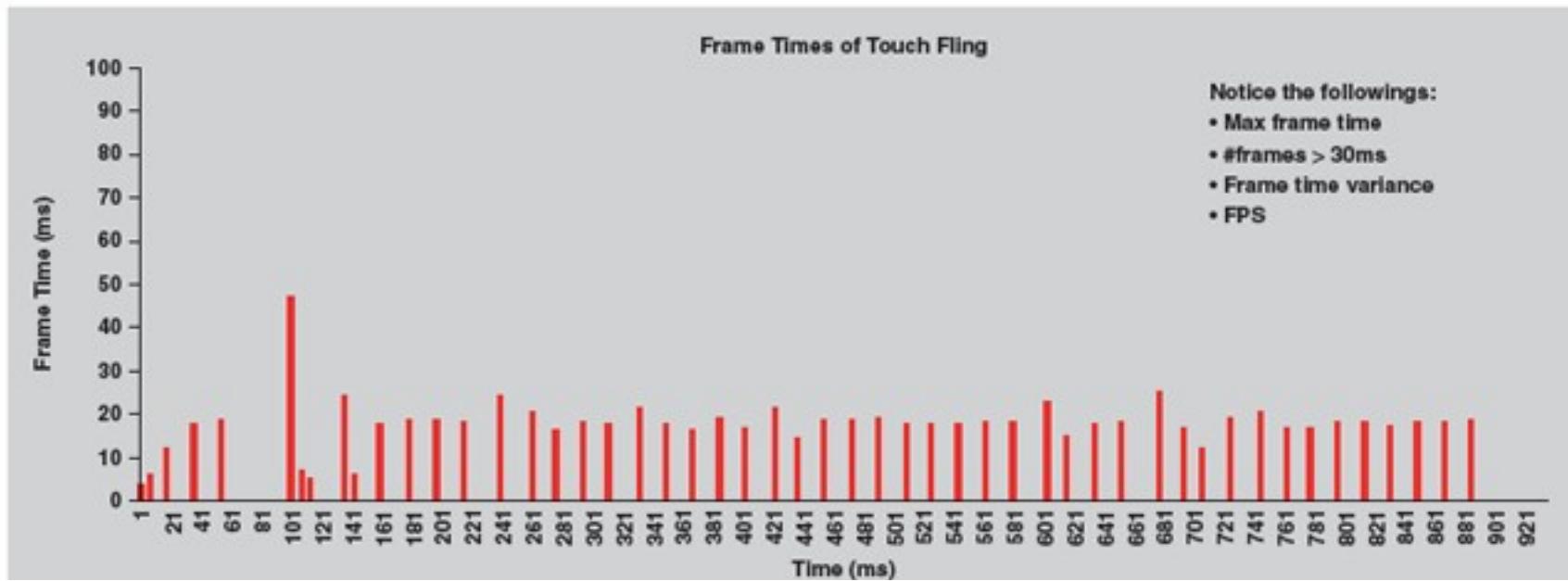
Другая проблема состоит в том, что FPS как ключевая величина гладкости взаимодействия не всегда отражает ощущения пользователя.

Например, при скроллинге изображения в приложении Gallery3D на устройстве Б мы видим ощутимые подтормаживания, а на устройстве А всё идет гладко, хотя FPS на нем ниже.

Чтобы понять, в чем проблема, мы нанесли период отрисовки фреймов на ось времени.

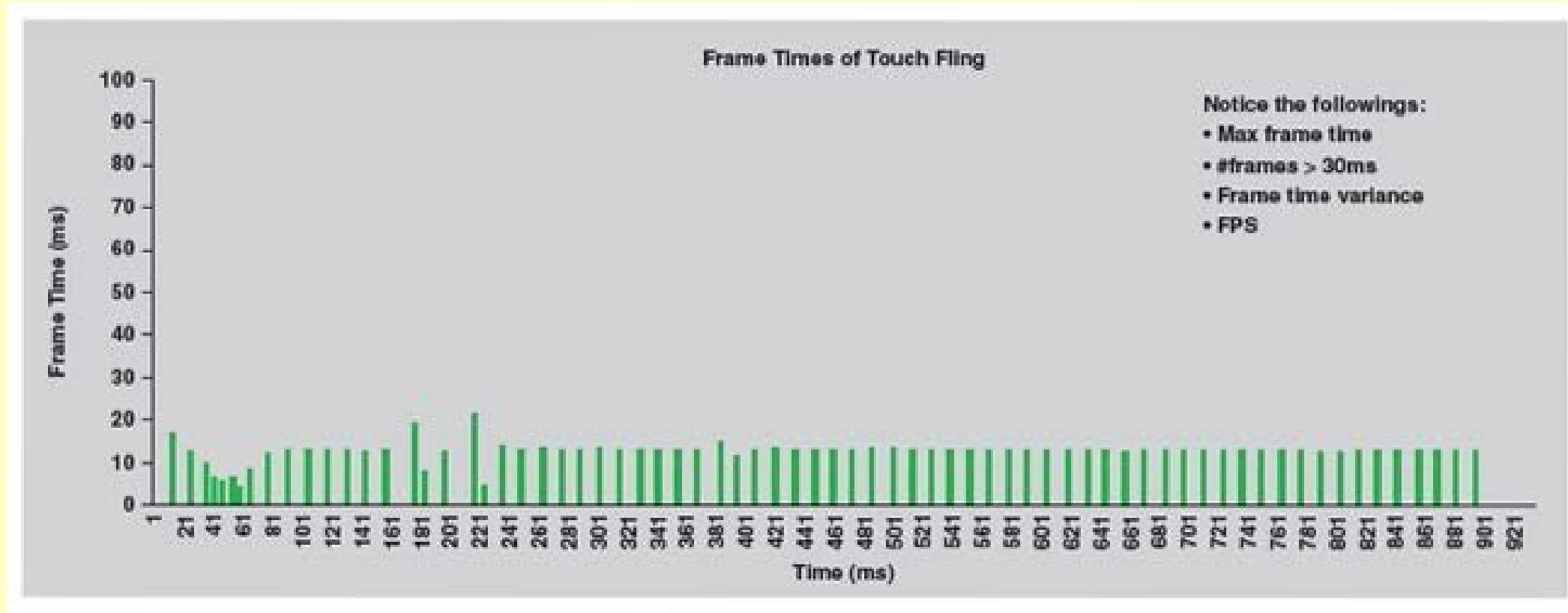
Теперь, наверное, причина видна всем: кроме FPS как такого, надо учитывать его стабильность, то есть вводить метрику максимального отклонения от средней величины.

Фреймрейты в приложении Gallery3D на устройствах A и Б



В качестве сравнения приведем график фреймрейта устройства Б после его оптимизации.

Как видим, средний FPS почти не изменился, чего не скажешь об ощущениях пользователя.



Фреймрейт на устройстве Б после оптимизации

Не следует забывать, что ощущение пользователей – понятие субъективное.

В современной науке используется несколько методов отслеживания реакции пользователей: мониторинг движения глаз, пульса и т.д.

При создании программного обеспечения мы должны подходить к вопросу системно, используя все возможные методы анализа.

Разработав же систему метрик (скажем, уже упоминавшиеся чувствительность, плавность, логичность и точность) необходимо определить границы их комфортности для пользователя.

В таблице ниже приведены некоторые полученные опытным путем величины.

В таблице ниже приведены некоторые полученные опытным путем величины.

	Отлично	Хорошо	Допустимо
Время задержки	$\leq 100 \text{ ms}$	$\leq 200 \text{ ms}$	$\leq 500 \text{ ms}$
Графическая анимация	$\geq 120 \text{ fps}$	$\geq 60 \text{ fps}$	$\geq 30 \text{ fps}$
Воспроизведение видео	$\geq 60 \text{ fps}$	$\geq 30 \text{ fps}$	$\geq 20 \text{ fps}$

Очевидно, что эти данные носят статистический характер и должны применяться с оглядкой на человеческую натуру.

Основываясь на опыте разработки под Android, можно прийти к выводу, что пользовательская оптимизация (ПО) приложений во многом схожа с оптимизацией распараллеливанием, только более сложна в реализации по следующим причинам:

- ПО затрагивает множество программных и аппаратных компонент, а также их взаимодействие;
- ПО вынуждена считаться с вопросами энергопотребления, поскольку это также влияет на ощущения пользователей;
- ПО оперирует жесткими временными рамками; приложение должно работать с комфортом пользователю скоростью, не быстрее и не медленее;
- ПО носит во многом субъективный характер, и многое зависит от чутья разработчика.

Управление энергопотреблением

Энергоэффективность всегда была головной болью для разработчиков мобильных ОС. Прожорливость приложений постоянно растет, и прогресс в аккумуляторных технологиях за ней хронически не успевает.

Вот почему важность управления питанием все время возрастает, и для решения этой проблемы необходимо применять поистине глобальный подход.

За последнее десятилетие значительных успехов в области экономии энергии достигли мобильные процессоры.

Современные модели поддерживают технологии динамического изменения напряжения и частоты, таких как Enhanced Intel SpeedStep.

Со стороны ОС управлением режимами работы процессора занимаются специальные компоненты ядра, такие, например, как cpubr freq в Linux.

В настоящее время мы наблюдаем процесс перемещения передового фронта борьбы за энергоэффективность от процессоров (где уже сделано немало) к другим системам мобильных устройств. Например, внедрение динамического управления графическим процессором (подобного тому, что применяется в ЦПУ) позволяет в некоторых случаях экономить до 50% энергии.

Внимания заслуживают также системы ввода-вывода; повышение их интеллектуальности, способности самостоятельно выбирать оптимальный режим работы также положительно скажется на потреблении.

В нынешних ОС ситуация с энергопотреблением такова.

ОС Android исповедует принцип «гибкого саспенда».

Не имея средств управления рабочим энергопотреблением устройства, Android агрессивно пытается перевести систему в состояние саспенда, если в ней не происходит ничего интересного, что определяется отсутствием блокировок (wakelock).

Windows 8,10, предлагает принципиально новое состояние устройства, названное «подключенным ждущим режимом».

В отличие от традиционного ждущего режима S3, при котором приостанавливаются все системные процессы, здесь система продолжает работать в чрезвычайно экономном режиме, позволяя, например, принимать e-mail.

Подключенный ждущий режим реализован аппаратно в процессоре и программно в ядре системы.

Корректность работы приложений с точки зрения энергопотребления остается ахиллесовой пятой обоих описанных подходов к сбережению.

Недавние исследования показали, что бесплатные приложения Android потребляют 75% энергии впустую, показывая рекламу в свернутом режиме и не отдавая блокировку.

То же самое справедливо и для Windows 8,10, где даже одно приложение, написанное неверно с точки зрения энергоэффективности, не позволит всей системе уйти в подключенный ждущий режим.

В настоящее время не существует четкого понимания, как бороться с такого рода «кривыми» приложениями.

Открытость

Другой важной отличительной чертой мобильной ОС является ее открытость. Под открытостью мы понимаем меру свободы в использовании, распространении, настройки и усовершенствовании ОС для своих нужд.

Существует отдельное исследование, посвященное открытости ОС с точки зрения разработчика; здесь же мы рассматриваем ее в рамках экосистемы, то есть всех сторон, так или иначе связанных с эксплуатацией этой ОС.

Еще совсем недавно большинство телефонов имели внутри себя закрытое ПО, куда не имели доступ сторонние разработчики; пользователям же приходилось довольствоваться встроенным инструментарием.

В процессе эволюции появились смартфоны с операционными системами, допускающими установку стороннего ПО, которое взаимодействовало с ОС посредством API; разработчикам были предоставлены соответствующие инструменты программирования (SDK).

Хорошим примером ОС подобного рода является Apple iOS.

Большую свободу для всей экосистемы предоставляют ОС с открытым кодом, как, например, Android; преимущества открытого кода может почувствовать даже конечный пользователь, не имеющий отношения к программированию – они, например, в количестве производителей, использующих эту ОС и, в конечном счете, количестве моделей.

Поддержка облачных технологий

Облачные технологии находят все более широкое распространение в мобильных ОС; в большинстве своем, приложения, их использующие, представляют собой веб-сайты, открывающиеся в браузере или веб-приложения.

Очень часто приложения созданы на HTML5, поэтому реализация данной технологии в мобильной ОС находится в центре внимания ее разработчиков.

Сайт html5test.com оценивает количественно поддержку HTML на различных платформах, приведем результаты в виде таблицы .

Браузер	Платформа	Оценка + бонус
Tizen 2		492 + 16
BlackBerry 10	BlackBerry Q10 или Z10	485 + 11
Dolphin Engine	Android 10 или выше	469 + 3
Opera Mobile 14	Android 10	448 + 11
Tizen 1		426 + 16
Firefox Mobile	Различные платформы	422 + 14
Chrome	Android 10	417 + 11

Браузеры сторонних производителей находятся в менее выигрышной ситуации по сравнению со встроенными в ОС, поскольку не имеют такого контроля над системой и процессом ее разработки.

В идеале, браузер должен работать в своей собственной среде, к этому стремятся все разработчики; про Firefox OS сейчас слышали уже многие, а вот на видео можно посмотреть превью Opera OS.

*Отметим, впрочем, что независимые разработчики по-прежнему легко удерживаются в топе, очередной прорыв – китайский браузер *Dolphin Engine* .*

Веб приложения, то есть локальные программы, использующие веб-технологии, требуют поддержки со стороны мобильной ОС (и не только) в виде среды исполнения, фреймворков и средств разработки.

- Среда исполнения обеспечивает работоспособность приложения. Свое происхождение она ведет от браузера, однако более интегрирована в исполняемую среду самой ОС;
- Веб фреймворк предоставляет богатые функционально библиотеки для разработки приложений, примером могут служить jQueryMobile или Sencha;
- Средства разработки должны быть гибкими, чтобы разработчики могли использовать их для своих разнообразных нужд.

HTML5 декларируется как кросс-платформенный стандарт, однако в реальности поддержка HTML5 у разных платформ не одинакова и процесс ее стандартизации сейчас в самом разгаре.

Для преодоления несовместимости создан фреймворк PhoneGap, который поддерживается всеми основными мобильными ОС.

Желание иметь общий для всех HTML5 есть у всех участников экосистемы, однако реализовать его не так-то просто, поскольку разработчики мобильных ОС постоянно внедряют в него свои собственные идеи.

Проблема производительности беспокоит всех разработчиков, имеющих дело с HTML5. На наш взгляд, наиболее актуальные аспекты оптимизации под мобильные операционные системы таковы:

- Аппаратное ускорение. Графика и видео должны иметь аппаратное ускорение;
- Поддержка многопоточности. Web Worker должен непременно поддерживаться в HTML5;
- Оптимизация движка JavaScript. JIT (Just In Time) включена во всех основных реализациях JavaScript;
- Поддержка нативных или гибридных приложений – возможность использования существующих системных библиотек будет еще одним подходом при создании веб-приложений

Подводя итог всему сказанному, еще раз отметим, что несмотря на некоторые различия в подходах, все мобильные ОС развиваются в одном направлении, в какой-то степени сближаясь друг с другом. Думается, что эта тенденция сохранится и в дальнейшем, что идет только на руку как пользователям, так и разработчикам приложений.

Android

Android — самая популярная операционная система и платформа для приложений, насчитывающая больше двух миллиардов активных пользователей.

На ней работают совершенно разные устройства, от «интернета вещей» и умных часов до телевизоров, ноутбуков и автомобилей, но чаще всего Android используют на смартфонах и планшетах.

Android — свободный и открытый проект.

Большинство исходного кода (который можно найти на <https://source.android.com>) распространяется под свободной лицензией Apache 2.0.

Публичная бета Android вышла в 2007 году, а первая стабильная версия - в 2008, с тех пор мажорные релизы выходят примерно раз в год.

Последняя на данный момент стабильная версия Android 12

Как и в других Linux-системах, ядро Linux в Android обеспечивает такие низкоуровневые вещи, как управление памятью, защиту данных, поддержку мультипроцессности, многопоточности , работу командного процессора, обработку системных вызовов.

Но за несколькими исключениями - вы не найдёте в Android других привычных компонентов GNU/Linux-систем: здесь нет ничего от проекта GNU, не используется X.Org, ни даже systemd.

Все эти компоненты заменены аналогами, более приспособленными для использования в условиях ограниченной памяти, низкой скорости процессора и минимального потребления энергии - таким образом, Android больше похож на встраиваемую (embedded) Linux-систему, чем на GNU/Linux.

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

Само ядро Linux в Android тоже немного модифицировано:
было добавлено несколько небольших компонентов, в том числе

- ashmem (anonymous shared memory),
- Binder driver (часть большого и важного фреймворка Binder),
- wakelocks (управление спящим режимом)
- low memory killer.

Исходно они представляли собой патчи к ядру, но их код был довольно
быстро добавлен назад в upstream-ядро.

Тем не менее, вы не найдёте их в «обычном линуксе»: большинство других
дистрибутивов отключают эти компоненты при сборке.

В качестве libc (стандартной библиотеки языка C) в Android используется не GNU C library (glibc), а собственная минималистичная реализация под названием bionic.

Она оптимизирована для встраиваемых (embedded) систем и значительно быстрее, меньше и менее требовательна к памяти, чем glibc, которая обросла множеством слоёв совместимости.

В Android есть оболочка командного процессора (shell) и множество стандартных для Unix-подобных систем команд/программ.

Во встраиваемых системах для этого обычно используется пакет Busybox, реализующий функциональность многих команд в одном исполняемом файле.

В Android используется его аналог под названием Toybox.

Как и в «обычных» дистрибутивах Linux (и в отличие от встраиваемых систем), основным способом взаимодействия с системой является графический интерфейс, а не командная строка.

Тем не менее, «добраться» до командной строки очень просто — достаточно запустить приложение-эмулятор терминала.

По умолчанию он обычно не установлен, но его легко, например, скачать из Play Store (Terminal Emulator for Android, Material Terminal, Termux).

Во многих «продвинутых» дистрибутивах Android — таких, как LineageOS (бывший CyanogenMod) — эмулятор терминала предустановлен.



≡ Window 1 ▾

+ ⌂ :

```
bullhead:/ $ uname  
Linux  
bullhead:/ $ ls /system | wc -l  
17  
bullhead:/ $ █
```

1 2 3 4 5 6 7 8 9 0

q w e r t y u i o p

a s d f g h j k l

▀ z x c v b n m ✕

?123 , ⓘ English

.



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

```
user@desktop-linux$ adb shell
```

```
android$ uname
```

```
Linux
```

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

Кроме того, раньше для реализации WebView использовался браузерный движок WebKit, но начиная с версии 7.0 вместо этого используется установленное приложение Chrome (или другое; список приложений, которым разрешено выступать в качестве WebView provider, конфигурируется на этапе компиляции системы).

Внутри себя Chrome тоже использует основанный на WebKit движок Blink, но в отличие от системной библиотеки, Chrome обновляется через Play Store. Таким образом, все приложения, использующие WebView, автоматически получают последние улучшения и исправления уязвимостей.

Как легко заметить, использование Android принципиально отличается от использования «обычного Linux» — вам не нужно открывать и закрывать приложения, вы просто переключаетесь между ними, как будто все приложения запущены всегда.

Действительно, одна из уникальных особенностей Android в том, что приложения не контролируют напрямую процесс, в котором они запущены. Давайте поговорим об этом подробнее.

Основная единица в Unix-подобных системах — процесс.

И низкоуровневые системные сервисы, и отдельные команды в shell'e, и графические приложения — это процессы.

В большинстве случаев процесс представляет собой чёрный ящик для остальной системы — другие компоненты системы не знают и не заботятся о его состоянии.

Процесс начинает выполняться с вызова функции `main()` (на самом деле `_start`), и дальше реализует какую-то свою логику, взаимодействуя с остальной системой через системные вызовы и простейшее межпроцессное общение (IPC).

Поскольку Android тоже Unix-подобен, всё это верно и для него, но в то время как низкоуровневые части — на уровне Unix — оперируют понятием процесса, на более высоком уровне — уровне Android Framework — основной единицей является приложение.

Приложение — не чёрный ящик: оно состоит из отдельных компонентов, хорошо известных остальной системе.

У приложений Android нет функции `main()`, нет одной точки входа. Вообще, Android максимально абстрагирует понятие приложение запущено как от пользователя, так и от разработчика.

Конечно, процесс приложения нужно запускать и останавливать, но Android делает это автоматически.

Разработчику предлагается реализовать несколько отдельных компонентов, каждый из которых обладает своим собственным жизненным циклом.

Binder

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

Этот механизм — Binder.

Разработка Binder началась в Be Inc. (для BeOS), затем он был портирован на Linux и открыт.

Основной разработчик Binder, Dianne Hackborn, была и остаётся одним из основных разработчиков Android.

За время разработки Android Binder был полностью переписан.

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

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

Низкоуровневые части Binder оперируют в терминах объектов, которые могут пересыпаться между процессами.

При этом используется подсчёт ссылок (reference-counting) для автоматического освобождения неиспользуемых общих ресурсов и уведомление о завершении удалённого процесса (link-to-death) для освобождения ресурсов внутри процесса.

Высокоуровневые части Binder работают в терминах интерфейсов, сервисов и прокси-объектов.

Описание интерфейса, предоставляемого программой другим программам, записывается на специальном языке AIDL (Android Interface Definition Language), внешне очень похожем на объявление интерфейсов в Java.

По этому описанию автоматически генерируется настоящий Java-интерфейс, который потом может использоваться и клиентами, и самим сервисом.

Кроме того, по .aidl-файлу автоматически генерируются два специальных класса:

- Proxy (для использования со стороны клиента)
- Stub (со стороны сервиса), реализующие этот интерфейс.

Для Java-кода в процессе-клиенте прокси-объект выглядит как обычный Java-объект, который реализует наш интерфейс, и этот код может просто вызывать его методы.

При этом сгенерированная реализация прокси-объекта автоматически сериализует переданные аргументы, общается с процессом-сервисом через `libbinder`, десериализует переданный назад результат вызова и возвращает его из Java-метода.

`Stub` работает наоборот: он принимает входящие вызовы через `libbinder`, десериализует аргументы, вызывает абстрактную реализацию метода, сериализует возвращаемое значение и передаёт его процессу-клиенту.

Соответственно, для реализации сервиса программисту достаточно реализовать абстрактные методы в унаследованном от `Stub` классе.

Такая реализация `Binder` на уровне Java позволяет большинству кода использовать прокси-объект, вообще не задумываясь о том, что его функциональность реализована в другом процессе.

Для обеспечения полной прозрачности `Binder` поддерживает вложенные и рекурсивные межпроцессные вызовы.

Более того, использование `Binder` со стороны клиента выглядит совершенно одинаково, независимо от того, расположена ли реализация используемого сервиса в том же или в отдельном процессе.

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

Binder широко используется в Android для реализации системных сервисов (например, пакетного менеджера и буфера обмена), но детали этого скрыты от разработчика приложений высокоуровневыми классами в Android Framework, такими как Activity, Intent и Context.

Приложения могут также использовать Binder для предоставления друг другу собственных сервисов — например, приложение Google Play Services вообще не имеет собственного графического интерфейса для пользователя, но предоставляет разработчикам других приложений возможность пользоваться сервисами Google Play.

ART

В отличие от большинства других высокоуровневых языков, программы на Java не распространяются в виде исходного кода, а компилируются в промежуточный формат (байт-код, byte-code), который представляет собой исполняемый бинарный код для специального процессора.

Хотя делаются попытки создать физический процессор, который исполнял бы Java-байт-код напрямую, в подавляющем большинстве случаев в качестве такого процессора используется эмулятор — Java virtual machine (JVM).

Обычно используется реализация от Oracle/OpenJDK под названием HotSpot.

В Android используется собственная реализация под названием Android Runtime (ART), специально оптимизированная для работы на мобильных устройствах.

В старых версиях Android (до 5.0 Lollipop) вместо ART использовалась другая реализация под названием Dalvik.

И в Dalvik, и в ART используется собственный формат байткода и собственный формат файлов, в которых хранится байт-код — DEX (Dalvik executable).

В отличие от class-файлов в «обычной джаве», весь Java-код приложения обычно компилируется в один DEX-файл `classes.dex`.

При сборке Android-приложения Java-код сначала компилируется обычным компилятором Java в class-файлы, а потом конвертируется в DEX-файл специальной утилитой (возможно и обратное преобразование).

И HotSpot, и Dalvik, и ART дополнительно оптимизируют выполняемый код. Все три используют just-in-time compilation (JIT), то есть во время выполнения компилируют байт-код в куски полностью нативного кода, который выполняется напрямую.

Кроме очевидного выигрыша в скорости, это позволяет оптимизировать код для выполнения на конкретном процессоре, не отказываясь от полной переносимости байт-кода.

Кроме того, ART может компилировать байткод в нативный код заранее, а не во время выполнения (*ahead-of-time compilation*) — причём система автоматически планирует эту компиляцию на то время, когда устройство не используется и подключено к зарядке (например, ночью).

При этом ART учитывает данные, собранные профилировщиком во время предыдущих запусков этого кода (*profile-guided optimization*).

Такой подход позволяет дополнительно оптимизировать код под специфику работы конкретного приложения и даже под особенности использования этого приложения именно этим пользователем.

В результате всех этих оптимизаций производительность Java-кода на Android не сильно уступает производительности низкоуровневого кода (на C/C++), а в некоторых случаях и превосходит её.

Java-байткод, в отличие от обычного исполняемого кода, использует объектную модель Java — то есть в байткоде явно записываются такие вещи, как классы, методы и сигнатуры.

Это делает возможной компиляцию других языков в Java-байткод, что позволяет написанным на них программам исполняться на виртуальной машине Java и быть в той или иной степени совместимыми (*interoperable*) с Java.

Существуют как JVM-реализации независимых языков — например, JPython для Python, JRuby для Ruby, Rhino для JavaScript и dialect Lisp Clojure — так и языки, исходно разработанные для компиляции в Java-байткод и выполнения на JVM, самые известные из которых — Groovy, Scala и Kotlin.

Самый новый из них, Kotlin, специально разработанный для идеальной совместимости с Java и обладающий гораздо более приятным синтаксисом (похожим на Swift), поддерживается Google как официальный язык разработки под Android наравне с Java.

Несмотря на все преимущества Java, в некоторых случаях всё-таки желательно использовать низкоуровневый язык — например, для реализации критичного по производительности компонента, такого как браузерный движок, или чтобы использовать существующую нативную библиотеку.

Java позволяет вызывать нативный код через Java Native Interface (JNI), и Android предоставляет специальные средства для нативной разработки — Native Development Kit (NDK), в который входят в том числе заголовочные файлы, компилятор (Clang), отладчик (LLDB) и система сборки.

Хотя NDK в основном ориентирован на использование C/C++, с его помощью можно писать под Android и на других языках — в том числе Rust, Swift, Python, JavaScript и даже Haskell. Больше того, есть даже возможность портировать iOS-приложения (написанные на Objective-C или Swift) на Android практически без изменений

Пакеты

Архитектура Android построена вокруг приложений.

Именно приложения играют ключевую роль в устройстве многих частей системы, именно для гармоничного взаимодействия приложений выстроена модель activity и intent'ов, именно на изоляции приложений основана модель безопасности Android.

И если оркестрированием взаимодействия компонентов приложений занимается activity manager, то за установку, обновление и управление правами приложений отвечает package manager (пакетный менеджер — в shell его можно вызвать командой pm).

Как подсказывает само название «пакетный менеджер», на этом уровне приложения часто называются пакетами.

Пакеты распространяются в формате APK (Android package) — специальных zip-архивов.

У каждого пакета есть имя (также известное как application ID), которое уникально идентифицирует это приложение (но не его конкретную версию — наоборот, имена разных версий пакета должны совпадать, иначе они будут считаться отдельными пакетами).

Имена пакетов принято записывать в нотации обратного DNS-имени — например, приложение YouTube использует имя пакета com.google.android.youtube.

Часто имя пакета совпадает с пространством имён, использующимся в его Java-коде, но Android этого не требует (к тому же, APK-файлы приложений обычно включают и сторонние библиотеки, пространство имён которых, естественно, не имеет вообще ничего общего с именами пакетов, которые их используют).

Каждый APK при сборке должен быть подписан разработчиком с использованием цифровой подписи.

Android проверяет наличие этой подписи при установке приложения, а при обновлении уже установленного приложения дополнительно сравнивает публичные ключи, которыми подписаны старая и новая версия; они должны совпадать, что гарантирует, что новая версия была создана тем же разработчиком, что и старая.

(Если бы этой проверки не было, злоумышленник мог бы создать пакет с таким же именем, как и у существующего приложения, убедить пользователя установить его, «обновляя» приложение, и получить доступ к данным этого приложения.)

Само обновление пакета представляет собой установку его новой версии вместо старой с сохранением данных и полученных от пользователя разрешений.

Можно и «откатывать» (downgrade) приложения до более старых версий, но при этом по умолчанию Android стирает сохранённые новой версией данные, поскольку старая версия может быть не способна работать с форматами данных, которые использует новая версия.

Как я уже говорил, обычно код каждого приложения выполняется под собственным Unix-пользователем (UID), что и обеспечивает их взаимную изоляцию.

Несколько приложений могут явно попросить Android использовать для них общий UID, что позволит им получать прямой доступ к файлам друг друга и даже, при желании, запускаться в одном процессе.

Хотя обычно одному пакету соответствует один файл APK, Android поддерживает пакеты, состоящие из нескольких APK (это называется раздельными APK, или *split APK*).

Это лежит в основе таких «магических» возможностей Android, как динамическая загрузка дополнительных модулей приложения (*dynamic feature modules*) и Instant Run в Android Studio (автоматическое обновление кода запущенного приложения без его полной переустановки и, во многих случаях, даже без перезапуска).

Файловая система

Устройство файловой системы — один из самых важных и интересных вопросов в архитектуре операционной системы, и устройство файловой системы в Android — не исключение.

Интерес представляет, во-первых, то, какие файловые системы используются, то есть в каком именно формате содержимое файлов сохраняется на условный диск (в случае Android это обычно flash-память и SD-карты) и как обеспечивается поддержка этого формата со стороны ядра системы.

Ядро Linux, используемое в Android, в той или иной степени поддерживает большое количество самых разных файловых систем — от используемых в Windows FAT и NTFS и используемых в Darwin печально известной HFS+ и современной APFS — до сетевой 9pfs из Plan 9. Есть и много «родных» для Linux файловых систем — например, Btrfs и семейство ext.

Стандартом де-факто для Linux уже долгое время является ext4, используемая по умолчанию большинством популярных дистрибутивов Linux.

Поэтому нет ничего неожиданного в том, что именно она и используется в Android.

В некоторых сборках (и некоторыми энтузиастами) также используется F2FS (Flash-Friendly File System), оптимизированная специально для flash-памяти (впрочем, с её преимуществами всё не так однозначно).

Во-вторых, интерес представляет так называемый *filesystem layout* — расположение системных и пользовательских папок и файлов в файловой системе.

Filesystem layout в «обычном Linux» заслуживает более подробного описания (которое можно найти, например, по этой [ссылке](#)); я упомяну здесь только несколько наиболее важных директорий:

- /home хранит домашние папки пользователей; здесь же, в различных скрытых папках (.var, .cache, .config и других), программы хранят свои настройки, данные и кэш, специфичные для пользователя,
- /boot хранит ядро Linux и образ initramfs (специальной загрузочной файловой системы),
- /usr (логичнее было бы назвать /system) хранит основную часть собственно системы, в том числе библиотеки, исполняемые файлы, конфигурационные файлы, а также ресурсы — темы для интерфейса, значки, содержимое системного мануала и т.п.,
- /etc (логичнее было бы назвать /config) хранит общесистемные настройки,
- /dev хранит файлы устройств и другие специальные файлы (например, сокет /dev/log),
- /var хранит изменяемые данные — логи, системный кэш, содержимое баз данных и т.п.

Android использует похожий, но заметно отличающийся filesystem layout.

Вот несколько самых важных из его частей:

- /data хранит изменяемые данные,
- ядро и образ initramfs хранятся на отдельном разделе (partition) flash-памяти, который не монтируется в основную файловую систему,
- /system соответствует /usr и хранит систему,
- /vendor — аналог /system, предназначенный для файлов, специфичных для этой сборки Android, а не входящих в «стандартный» Android,
- /dev, как и в «обычном Linux», хранит файлы устройств и другие специальные файлы.

Наиболее интересные из этих директорий – /data и /system.

Содержимое /system описывает систему и содержит большинство составляющих её файлов. /system располагается на отдельном разделе flash-памяти, который по умолчанию монтируется в режиме read-only; обычно данные на нём изменяются только при обновлении системы.

/data также располагается на отдельном разделе и описывает изменяемое состояние конкретного устройства, в том числе пользовательские настройки, установленные приложения и их данные, кэши и т.п.

Очистка всех пользовательских данных, так называемый factory reset, при такой схеме заключается просто в очистке содержимого раздела data; нетронутая система остаётся установлена в разделе system.

```
# mount | grep /system
```

```
/dev/block/mmcblk0p14 on /system type ext4 (ro,seclabel,relatime,data=ordered)
```

```
# mount | grep /data
```

```
/dev/block/mmcblk0p24 on /data type ext4  
(rw,seclabel,nosuid,nodev,noatime,noauto_da_alloc,data=writeback)
```

Приложения — а именно, их APK, файлы odex (скомпилированный ahead-of-time Java-код) и ELF-библиотеки — устанавливаются в /system/app (для приложений, поставляемых с системой) или в /data/app (для установленных пользователем приложений).

Каждому предустановленному приложению при создании сборки Android выделяется папка с именем вида /system/app/Terminal, а для устанавливаемых пользователем приложений при установке создаются папки, имена которых начинаются с их имени пакета. Например, приложение YouTube сохраняется в папку с названием вроде /data/app/com.google.android.youtube-bkJAtUtbTuzvAioW-LEStg==/.

Суффикс в названии папок приложений — 16 случайных байт, закодированных в Base64.

Использование такого суффикса не позволяет другим приложениям «угадать» путь к приложению, о существовании которого им знать не следует.

В принципе, список установленных на устройстве приложений и путей к ним не является секретом — его можно получить через стандартные API — но в некоторых случаях (а именно, для Instant apps) на доступ к этим данным накладываются ограничения.

Этот суффикс служит и другой цели. Каждый раз при обновлении приложения новая APK устанавливается в папку с новым суффиксом, после чего старая папка удаляется.

До версии 8.0 Oreo в этом и состояло назначение суффиксов, и вместо случайных байт поочерёдно использовались -1 и -2 (например, /data/app/com.google.android.youtube-2 для YouTube).

Полный путь к папке приложения в /system/app или /data/app можно получить с помощью стандартного API или команды pm path org.example.packagename, которая выводит пути всех APK-файлов приложения.

```
# pm path com.android.egg  
package:/system/app/EasterEgg/EasterEgg.apk
```

Поскольку предустановленные приложения хранятся в разделе `system` (содержимое которого, напомню, изменяется только при обновлении системы), их невозможно удалить (вместо этого Android предоставляет возможность их «отключить»).

Тем не менее, поддерживается обновление предустановленных приложений — при этом для новой версии создаётся папка в `/data/app`, а поставляемая с системой версия остаётся в `/system/app`.

В этом случае у пользователя появляется возможность «удалить обновления» такого приложения, вернувшись к версии из `/system/app`.

Другая особенность предустановленных приложений — они могут получать специальные «системные» разрешения.

Например, сторонним приложениям не получить разрешения `DELETE_PACKAGES`, которое позволяет удалять другие приложения, `REBOOT`, позволяющего перезапустить систему, и `READ_FRAME_BUFFER`, позволяющего получить прямой доступ к содержимому экрана.

Эти разрешения имеют уровень защиты `signature`, то есть приложение, пытающееся получить доступ к ним, должно быть подписано тем же ключом, что и приложение или сервис, в котором они реализованы — в этом случае, поскольку эти разрешения реализованы системой, ключом, которым подписана сама сборка Android.

Для хранения изменяемых данных каждому приложению выделяется папка в /data/data (например, /data/data/com.google.android.youtube для YouTube).

Доступ к этой папке есть только у самого приложения — то есть только у UID, под которым запускается это приложение (если приложение использует несколько UID, или несколько приложений используют общий UID, всё может быть сложнее).

В этой папке приложения сохраняют настройки, кэш (в подпапках shared_prefs и cache соответственно) и любые другие нужные им данные:

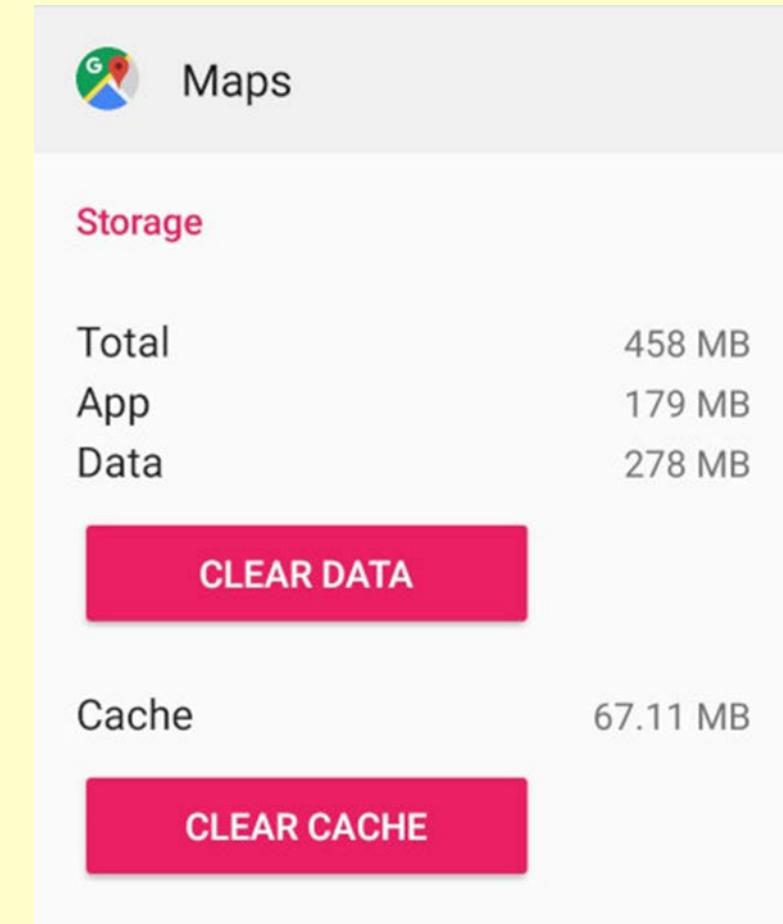
```
# ls /data/data/com.google.android.youtube/
cache code_cache databases files lib no_backup shared_prefs
# cat /data/data/com.google.android.youtube/shared_prefs/youtube.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="user_account">username@gmail.com</string>
    <boolean name="h264_main_profile_supported7.1.2" value="true" />
    <int name="last_manual_video_quality_selection_max" value="-2" />
    <...>
</map>
```

Система знает о существовании папки cache и может очищать её самостоятельно при нехватке места.

При удалении приложения вся папка этого приложения полностью удаляется, и приложение не оставляет за собой следов.

Альтернативно, и то, и другое пользователь может явно сделать в настройках:

Это выделяемое каждому приложению хранилище данных называется внутренним хранилищем (internal storage).



Кроме того, в Android есть и другой тип хранилища — так называемое внешнее хранилище (external storage — это название отражает изначальную задумку о том, что внешнее хранилище должно было располагаться на вставляемой в телефон внешней SD-карте).

По сути, внешнее хранилище играет роль домашней папки пользователя — именно там располагаются такие папки, как Documents, Download, Music и Pictures, именно внешнее хранилище открывают файловые менеджеры в качестве папки по умолчанию, именно к содержимому внешнего хранилища Android позволяет получить доступ компьютеру при подключении по кабелю.

```
# ls /sdcard
Alarms      Download      Podcasts
Android     Movies        Ringtones
Books       Music         Subtitles
DCIM        Notifications bluetooth
Documents   Pictures
```

В отличие от внутреннего хранилища, разделённого на папки отдельных приложений, внешнее хранилище представляет собой «общую зону»: к нему есть полный доступ у любого приложения, получившего соответствующее разрешение от пользователя. Это разрешение стоит запрашивать таким приложениям, как файловый менеджер; а большинству остальных приложений лучше использовать intent с действием ACTION_GET_CONTENT, предоставляя пользователю возможность самому выбрать нужный файл в системном файловом менеджере.

Многие приложения предпочитают сохранять и некоторые из своих внутренних файлов, имеющие большой размер (например, кэш загруженных изображений и аудиофайлов) во внешнем хранилище. Для этого Android выделяет приложениям во внешнем хранилище папки с названиями вида

Android/data/com.google.android.youtube. Самому приложению для доступа к такой папке не требуется разрешение на доступ ко всему внешнему хранилищу (поскольку в качестве владельца этой папки устанавливается его UID), но к этой папке может получить доступ любое другое приложение, имеющее такое разрешение, поэтому её, действительно, стоит использовать только для хранения публичных и некритичных данных.

При удалении приложения система удалит и его специальную папку во внешнем хранилище; но файлы, созданные приложениями во внешнем хранилище вне их специальной папки считаются принадлежащими пользователю и остаются на месте после удаления создавшего их приложения.

Исходно предполагалось, что внешнее хранилище действительно будет располагаться на внешней SD-карте, поскольку в то время объём SD-карт значительно превышал объём встраиваемой в телефоны памяти (в том же самом HTC Dream её было лишь 256 мегабайта, из которых на раздел data выделялось порядка 90 мегабайт).

С тех пор многие условия изменились; в современных телефонах часто нет слота для SD-карты, зато устанавливается огромное по мобильным меркам количество встроенной памяти (например, в Samsung Galaxy Note 9 её может быть до 512 гигабайт).

Поэтому в современном Android практически всегда и внутреннее, и внешнее хранилища располагаются во встроенной памяти.

Настоящий путь, по которому располагается внешнее хранилище в файловой системе, имеет форму /data/media/0 (для каждого пользователя устройства создаётся отдельное внешнее хранилище, и число в пути соответствует номеру пользователя). В целях совместимости до внешнего хранилища также можно добраться по путям /sdcard, /mnt/sdcard, /storage/self/primary, /storage/emulated/0, некоторым путям, начинающимся с /mnt/runtime/, и некоторым другим.

С другой стороны, у многих устройств всё-таки есть слот для SD-карты. Вставленную в Android-устройство SD-карту можно использовать как обычный внешний диск (не превращая её во внутреннее или внешнее хранилище системы) — сохранять на неё файлы, открывать хранящиеся на ней файлы, использовать её для перенесения файлов на другие устройства и т.п.

Кроме того, Android позволяет «заимствовать» SD-карту и разместить внутреннее и внешнее хранилище на ней (это называется заимствованным хранилищем — adopted storage).

При этом система переформатирует SD-карту и шифрует её содержимое — хранящиеся на ней данные невозможno прочесть, подключив её к другому устройству.

Загрузка

Традиционный подход к безопасности компьютерных систем ограничивается тем, чтобы защищать систему от программных атак.

Считается, что если у злоумышленника есть физический доступ к компьютеру, игра уже проиграна: он может получить полный доступ к любым хранящимся на нём данным.

Для этого ему достаточно, например, запустить на этом компьютере произвольную контролируемую им операционную систему, позволяющую ему обойти любые накладываемые «основной» системой ограничения прав, или напрямую подключить диск с данными к другому устройству.

При желании, злоумышленник может оставить компьютер в работоспособном состоянии, но пропатчить установленную на нём систему, установить произвольные бэкдоры, кейлоггеры и т.п.

Именно на защиту от программных атак ориентирована модель ограничения прав пользователей в Unix (и основанная на ней технология app sandbox в Android); само по себе ограничение прав в Unix никак не защищает систему от пользователя, пробравшегося в серверную и получившего физический доступ к компьютеру.

И если серьёзные многопользовательские сервера можно и нужно охранять от неавторизованного физического доступа, к персональным компьютерам — а тем более мобильным устройствам — такой подход просто неприменим.

Пытаться улучшать ситуацию с защитой от злоумышленника, получившего физический доступ к устройству, можно по двум направлениям:

- Во-первых, можно шифровать хранящиеся на диске данные, тем самым не позволяя злоумышленнику получить доступ к самим данным, даже если у него есть доступ к содержимому диска.
- Во-вторых, можно так или иначе ограничивать возможность загрузки на устройстве произвольных операционных систем, вынуждая злоумышленника проходить процедуры аутентификации и авторизации в установленной системе.

Именно с этими двумя направлениями защиты связана модель безопасной загрузки в Android.

Verified Boot

Процесс загрузки Android построен так, что он, с одной стороны, не позволяет злоумышленникам загружать на устройстве произвольную ОС, с другой стороны, может позволять пользователям устанавливать кастомизированные сборки Android (и другие системы).

Прежде всего, Android-устройства, в отличие от «десктопных» компьютеров, обычно не позволяют пользователю (или злоумышленнику) произвести загрузку со внешнего носителя; вместо этого сразу запускается установленный на устройстве bootloader (загрузчик). Bootloader — это относительно простая программа, в задачи которой (при загрузке в обычном режиме) входят:

- инициализация и настройка Trusted Execution Environment (например, ARM TrustZone),
- нахождение разделов встроенной памяти, в которых хранятся образы ядра Linux и initramfs,
- проверка их целостности и неприкосновенности (*integrity*) — в противном случае загрузка прерывается с сообщением об ошибке — путём верификации цифровой подписи производителя,
- загрузка ядра и initramfs в память и передача управления ядру.

Flashing, unlocking, fastboot и recovery

Кроме того, bootloader поддерживает дополнительную функциональность для обновления и переустановки системы.

Во-первых, это возможность загрузить вместо основной системы (Android) специальную минимальную систему, называемую *recovery*.

Версия *recovery*, устанавливаемая на большинство Android-устройств по умолчанию, очень минималистична и поддерживает только установку обновлений системы в автоматическом режиме, но многие энтузиасты Android устанавливают кастомную *recovery*.

Это делается путём использования второй «фичи» bootloader'a, направленной на обновление и переустановку системы — поддержки перезаписи (*flashing*) содержимого и структуры разделов по командам с подсоединённого по кабелю компьютера.

Для этого bootloader способен загружаться в ещё один специальный режим, который называют *fastboot mode* (или иногда просто *bootloader mode*), поскольку обычно для общения между компьютером и bootloader'ом в этом режиме используется протокол *fastboot* (и соответствующий ему инструмент *fastboot* из Android SDK со стороны компьютера).

Некоторые реализации bootloader'a используют другие протоколы.

В основном это касается устройств, выпускаемых компанией Samsung, где специальная реализация bootloader'a (Loke) общается с компьютером по собственному проприетарному протоколу (Odin).

Для работы с Odin со стороны компьютера можно использовать либо реализацию от самих Samsung (которая тоже называется Odin), либо свободную реализацию под названием Heimdall.

Конкретные детали зависят от реализации bootloader'a (то есть различаются в зависимости от производителя устройства), но во многих случаях установка recovery и сборок Android, подписанных ключом производителя устройства, просто работает без дополнительных сложностей:

```
$ fastboot flash recovery recovery.img
```

```
$ fastboot flash boot boot.img
```

```
$ fastboot flash system system.img
```

Таким образом можно вручную обновлять систему; а вот установить более старую версию не получится: функция, известная как защита от отката, не позволит bootloader'у загрузить более старую версию Android, чем была загружена в прошлый раз, даже если она подписана ключом производителя, поскольку загрузка старых версий открывает дорогу к использованию опубликованных уязвимостей, которые исправлены в более новых версиях.

Кроме того, многие устройства поддерживают разблокировку bootloader'a (unlocking the bootloader, также известную как OEM unlock) — отключение проверки bootloader'ом подписи системы и recovery, что позволяет устанавливать произвольные сборки того и другого (у части производителей это аннулирует гарантию).

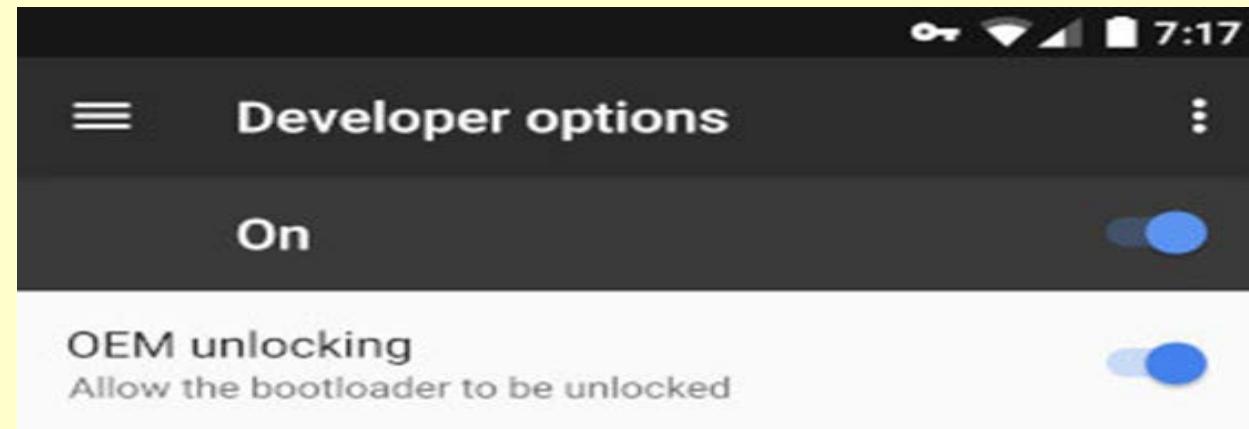
Именно так обычно устанавливаются такие популярные дистрибутивы Android, как LineageOS (бывший CyanogenMod), Paranoid Android, AOKP, OmniROM и другие.

Поскольку разблокировка bootloader'a всё-таки позволяет загрузить на устройстве собственную версию системы, в целях безопасности при разблокировке все пользовательские данные (с раздела data) принудительно удаляются.

Если систему переустанавливает сам пользователь, а не злоумышленник, после пере установки он может восстановить свои данные из бэкапа (например, из облачного бэкапа на серверах Google или из бэкапа на внешнем носителе), если злоумышленник — он получит работающую систему, но не сможет украсть данные владельца устройства.

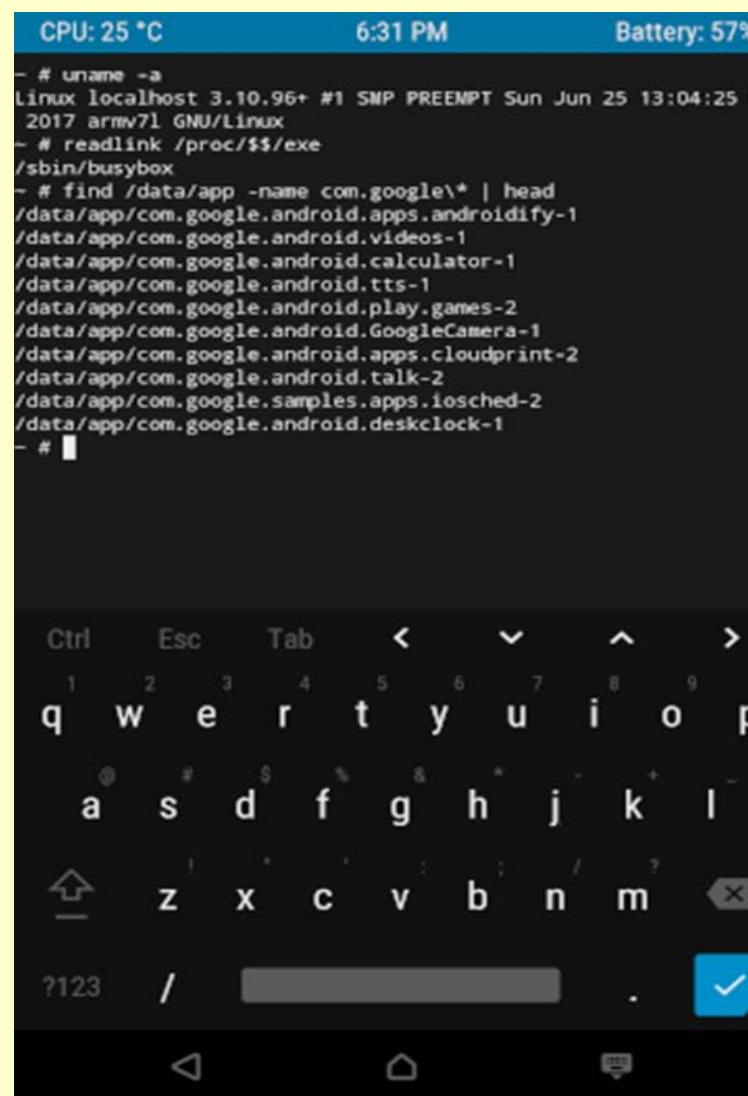
После установки предпочтаемых сборок recovery и системы bootloader стоит заблокировать обратно, чтобы снова защитить свои данные в случае попадания устройства в руки злоумышленников.

Для разблокировки bootloader'a может также потребоваться дополнительно разрешить её из настроек системы:



Популярная сторонняя recovery, которую устанавливают большинство «флешаголиков» (flashaholics) — TWRP (Team Win Recovery Project).

Она содержит тач-интерфейс и множество продвинутых «фич», в том числе возможность устанавливать части системы из сборок в виде zip-архивов, встроенную поддержку бэкапов и даже полноценный эмулятор терминала с виртуальной клавиатурой:



Шифрование диска

Современные версии Android используют пофайловое шифрование данных (file-based encryption).

Этот механизм основан на встроенной в ext4 поддержке шифрования, реализованной в ядре Linux (fscrypt), и позволяет системе зашифровывать различные части файловой системы различными ключами.

По умолчанию система шифрует большинство данных пользователя, расположенных на разделе data, с помощью ключа, который создаётся на основе пароля пользователя и не сохраняется на диск (credential encrypted storage).

Это означает, что при загрузке система должна попросить пользователя ввести свой пароль, чтобы вычислить с его помощью ключ для расшифровки данных.

Именно поэтому первый раз после включения устройства пользователя встречает требование ввести полный пароль или графический ключ, а не просто пройти аутентификацию, приложив палец к сканеру отпечатков.

В дополнение к credential encrypted storage в Android также используется device encrypted storage — шифрование ключом на основе данных, которые хранятся на устройстве (в том числе в Trusted Execution Environment).

Файлы, зашифрованные таким образом, система может расшифровать до того, как пользователь введёт пароль.

Это лежит в основе функции, известной как Direct Boot: система способна загружаться в некоторое работоспособное состояние и без ввода пароля; при этом приложения могут явно попросить систему сохранить (наименее приватную) часть своих данных в device encrypted storage, что позволяет им начинать выполнять свои базовые функции, не дожидаясь полной разблокировки устройства.

Например, Direct Boot позволяет будильнику срабатывать и до первого ввода пароля, что особенно полезно, если устройство непредвиденно перезагружается ночью из-за временного отключения питания или сбоя системы.

Root

Так называемый root-доступ — это возможность выполнять код от имени «пользователя root» (UID 0, также известного как суперпользователь).

Напомню, что root — это специально выделенный Unix-пользователь, которому — за несколькими интересными исключениями — разрешён полный доступ ко всему в системе, и на которого не распространяются никакие ограничения прав.

Как и большинство других современных операционных систем, Android спроектирован с расчётом на то, что обычному пользователю ни для чего не требуется использовать root-доступ.

В отличие от более закрытых операционных систем, пользователи которых называют разрушение наложенных на них ограничений буквально « побегом из тюрьмы », в Android прямо « из коробки », без необходимости получать root-доступ и устанавливать специальные сторонние « твики », есть возможность:

- устанавливать произвольные приложения — как из множества существующих магазинов приложений, так и произвольные APK из любых источников,
- выбирать приложения по умолчанию: веб-браузер, email-клиент, камеру, файловый менеджер, лончер, приложение для звонков, приложение для СМС, приложение для контактов и так далее (это работает за счёт красивой системы activity и intent'ов, которую я описал в прошлой статье),
- устанавливать и использовать пакеты значков (icon packs),
- получать прямой доступ к файловой системе, хранить в ней произвольные файлы,
- подключать устройство к компьютеру (или даже к другому Android-устройству) и напрямую передавать между ними файлы по кабелю,
- и даже, во многих дистрибутивах Android, настраивать цвета и шрифты системной темы.

Итак, для обычновенных задач, с которыми может столкнуться простой пользователь, root-доступ не нужен. В то же время использование root-доступа неизбежно сопряжено со многими проблемами с безопасностью (подробнее о них ниже), поэтому в большинстве случаев Android не позволяет пользователю работать от имени root. Приложения, в том числе эмуляторы терминалов, выполняются от имени своих ограниченных Unix-пользователей; a shell, который запускается при использовании команды adb shell, работает от имени специально для этого предназначенного Unix-пользователя shell.

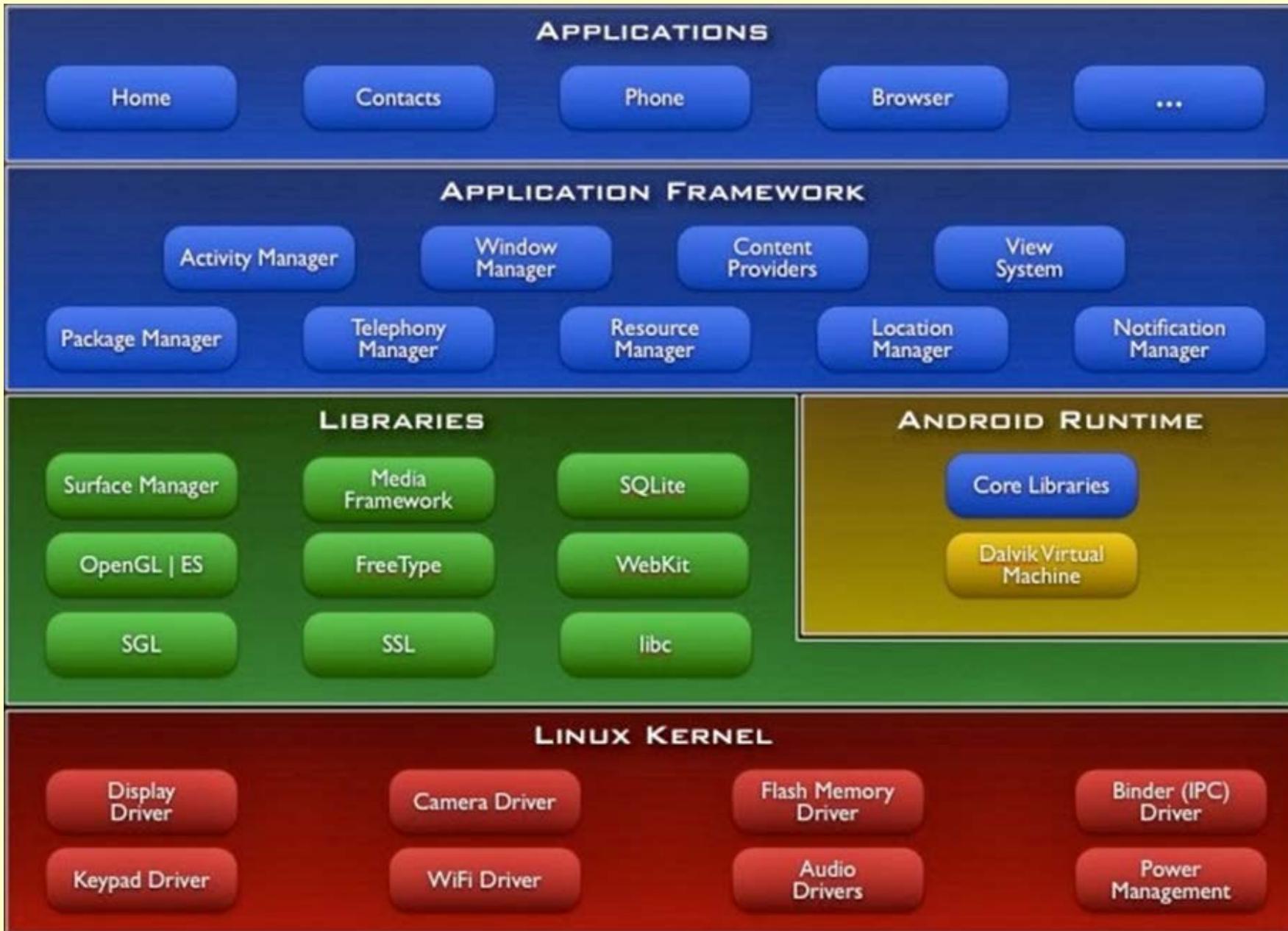
Тем не менее, бывает, что root доступен пользователю:

- Во-первых, root-доступ обычно разрешён на сборках Android для эмуляторов — поставляемых вместе с Android Studio виртуальных машин на базе QEMU, которые изображают реальные устройства и обычно используются разработчиками для отладки и тестирования приложений.

- Во-вторых, root-доступ включён по умолчанию во многих сторонних дистрибутивах Android (pre-rooted ROMs).
- В-третьих, при разблокированном bootloader'e root-доступ на любой сборке Android можно включить напрямую, просто установив исполняемый файл, реализующий команду su, через bootloader.
- Ну и наконец, в-четвёртых, на старых версиях системы, содержащих известные уязвимости, часто можно получить root-доступ, проэксплуатировав их (обычно для получения root-доступа эксплуатируются сразу несколько уязвимостей в разных слоях и компонентах системы).

Например, если система не обновлялась с первой половины 2016 года, для получения root-доступа можно воспользоваться получившей широкую известность уязвимостью Dirty Cow.

Классический рисунок представляющий архитектуру ОС Android:



Если кому-то сложно с английским, то на всякий случай то же самое по на русском:

Уровень приложений

Встроенные
приложения
(контакты, карты,
браузер и т. п.)

Сторонние
приложения

Приложения
разработчика

Фреймворк приложений

Навигационные
службы

Источники
данных

Оконный
менеджер

Менеджер
деятельностей

Менеджер
пакетов

Телефония

P2P/XMPP

Уведомления

Представления

Менеджер
ресурсов

Библиотеки

Графика
(OpenGL, SGL,
Free Type)

Мультимедиа

SSL & Webkit

libc

SQLite

Менеджер
поверхностей

Библиотеки
Android

Виртуальная
машина Dalvik

Рабочая среда Android

Linux-ядро

Драйверы
оборудования
(USB, экран,
Bluetooth и т. п.)

Управление
питанием

Управление
процессами

Управление
памятью

Если представить компонентную модель Android в виде некоторой иерархии, то в самом низу, как самая фундаментальная и базовая составляющая, будет располагаться ядро операционной системы (Linux Kernel).

Часто компонентную модель ещё называют программным стеком.

Действительно, это определение тут уместно, потому что речь идет о наборе программных продуктов, которые работают вместе для получения итогового результата.

Действия в этой модели выполняются последовательно, и уровни иерархии также последовательно взаимодействуют между собой.

LINUX KERNEL (ЯДРО ЛИНУКС)

Как известно, Андроид основан на несколько урезанном ядре ОС Linux и поэтому на этом уровне мы можем видеть именно его (версии x.x.x).

Оно обеспечивает функционирование системы и отвечает за безопасность, управление памятью, энергосистемой и процессами, а также предоставляет сетевой стек и модель драйверов.

Ядро также действует как уровень абстракции между аппаратным обеспечением и программным стеком.

LIBRARIES (БИБЛИОТЕКИ)

«Выше» ядра, как программное обеспечение промежуточного слоя, лежит набор библиотек (Libraries), предназначенный для обеспечения важнейшего базового функционала для приложений.

То есть именно этот уровень отвечает за предоставление реализованных алгоритмов для вышележащих уровней, поддержку файловых форматов, осуществление кодирования и декодирования информации (в пример можно привести мультимедийные кодеки), отрисовку графики и многое другое.

Библиотеки реализованы на C/C++ и скомпилированы под конкретное аппаратное обеспечение устройства, вместе с которым они и поставляются производителем в предустановленном виде.

Краткое описание некоторых из них:

- Surface Manager – в ОС Android используется композитный менеджер окон, наподобие Compiz (Linux), но более упрощенный.

Вместо того чтобы производить отрисовку графики напрямую в буфер дисплея, система посыпает поступающие команды отрисовки в закадровый буфер, где они накапливаются вместе с другими, составляя некую композицию, а потом выводятся пользователю на экран.

Это позволяет системе создавать интересные бесшовные эффекты, прозрачность окон и плавные переходы.

- Media Framework – библиотеки, реализованные на базе PacketVideo OpenCORE. С их помощью система может осуществлять запись и воспроизведение аудио и видео контента, а также вывод статических изображений. Поддерживаются многие популярные форматы, включая MPEG4, H.264, MP3, AAC, AMR, JPG и PNG.
- SQLite – легковесная и производительная реляционная СУБД, используемая в Android в качестве основного движка для работы с базами данных, используемыми приложениями для хранения информации

- OpenGL | ES –

3D библиотеки — используются для высокооптимизированной отрисовки 3D-графики, при возможности используют аппаратное ускорение.

Их реализации строятся на основе API OpenGL ES 1.0.

3D библиотеки которые используются для высоко оптимизированной отрисовки 3D-графики, при возможности используют аппаратное ускорение.

Их реализации строятся на основе API OpenGL ES 1.0.

OpenGL ES (OpenGL for Embedded Systems) – подмножество графического программного интерфейса OpenGL, адаптированное для работы на встраиваемых системах.

- FreeType – библиотека для работы с битовыми картами, а также для растеризации шрифтов и осуществления операций над ними. Это высококачественный движок для шрифтов и отображения текста.
 - LibWebCore – библиотеки известного шустрого браузерного движка WebKit, используемого также в десктопных браузерах Google Chrome и Apple Safari.
 - SGL (Skia Graphics Engine) – открытый движок для работы с 2D-графикой. Графическая библиотека является продуктом Google и часто используется в других их программах.
 - SSL - библиотеки для поддержки одноименного криптографического протокола.
 - Libc – стандартная библиотека языка C, а именно её BSD реализация, настроенная для работы на устройствах на базе Linux. Носит название Bionic.
- На этом же уровне располагается Android Runtime – среда выполнения. Ключевыми её составляющими являются набор библиотек ядра и виртуальная машина Dalvik. Библиотеки обеспечивают большую часть низкоуровневой функциональности, доступной библиотекам ядра языка Java.

ANDROID RUNTIME (СРЕДА ВЫПОЛНЕНИЯ АНДРОИД)

На этом же уровне располагается Android Runtime – среда выполнения.

Ключевыми её составляющими являются набор библиотек ядра (Core Libraries) и виртуальная машина Dalvik.

Библиотеки обеспечивают большую часть низкоуровневой функциональности, доступной библиотекам ядра языка Java.

Каждое приложение в ОС Android запускается в собственном экземпляре виртуальной машины Dalvik.

Таким образом, все работающие процессы изолированы от операционной системы и друг от друга.

И вообще, архитектура Android Runtime такова, что работа программ осуществляется строго в рамках окружения виртуальной машины.

Благодаря этому осуществляется защита ядра операционной системы от возможного вреда со стороны других её составляющих.

Поэтому код с ошибками или вредоносное ПО не смогут испортить Android и устройство на его базе, когда сработают.

Такая защитная функция, наряду с выполнением программного кода, является одной из ключевых для надстройки Android Runtime.

Dalvik полагается на ядро Linux для выполнения основных системных низкоуровневых функций, таких как, безопасность, потоки, управление процессами и памятью.

Вы можете также писать приложения на C/C++, которые будут работать непосредственно на базовом уровне ОС Linux. Хотя такая возможность и существует, необходимости в этом нет никакой.

Если для приложения важны присущие C/C++ скорость и эффективность работы, Android предоставляет доступ к нативной среде разработки (NDK – Native Development Kit).

Она позволяет разрабатывать приложения на C/C++ с использованием библиотек `libc` и `libm`, а также обеспечивает нативный доступ к OpenGL.

Доступ к устройствам и системным службам Android осуществляется через виртуальную машину Dalvik, которая считается промежуточным слоем.

Благодаря использованию Dalvik для выполнения кода программы разработчики получают в свое распоряжение уровень абстракции, который позволяет им не беспокоиться об особенностях конструкции того или иного устройства.

Виртуальная машина Dalvik может выполнять программы в исполняемом формате DEX (Dalvik Executable).

Данный формат оптимизирован для использования минимального объема памяти.

Исполняемый файл с расширением .dex создается путем компиляции классов Java с помощью инструмента dx, входящего в состав Android SDK.

При использовании IDE Eclipse и плагина ADT (Android Development Tools) компиляция классов Java в формат .dex происходит автоматически

Как было сказано выше, инструмент dx из Android SDK компилирует приложения, написанные на Java, в исполняемый формат (dex) виртуальной машины Dalvik.

Помимо непосредственно исполняемых файлов, в состав приложения Android входят прочие вспомогательные компоненты (такие, например, как файлы с данными и файлы ресурсов).

SDK упаковывает все необходимое для установки приложения в файл с расширением .apk (Android package).

Весь код в одном файле .apk считается одним приложением и этот файл используется для установки данного приложения на устройствах с ОС Android.

APPLICATION FRAMEWORK (КАРКАС ПРИЛОЖЕНИЙ)

Уровнем выше располагается Application Framework, иногда называемый уровнем каркаса приложений.

Именно через каркасы приложений разработчики получают доступ к API, предоставляемым компонентами системы, лежащими ниже уровнем.

Кроме того, благодаря архитектуре фреймворка, любому приложению предоставляются уже реализованные возможности других приложений, к которым разрешено получать доступ.

В базовый набор сервисов и систем, лежащих в основе каждого приложения и являющихся частями фреймворка, входят:

- Activity Manager – менеджер Активностей, который управляет жизненными циклами приложений, сохраняет данные об истории работы с Активностями, а также предоставляет систему навигации по ним.
- Package Manager – менеджер пакетов, управляет установленными пакетами на вашем устройстве, отвечает за установку новых и удаление существующих.
- Window Manager – менеджер окон, управляет окнами, и предоставляет для приложений более высокий уровень абстракции библиотеки Surface Manager.
- Telephony Manager – менеджер телефонии, содержит API для взаимодействия с возможностями телефонии (звонки, смс и т.п.)

- Content Providers – контент-провайдеры, управляют данными, которые одни приложения открывают для других, чтобы те могли их использовать для своей работы.
- Resource Manager – менеджер ресурсов, обеспечивает доступ к ресурсам без функциональности (не несущими кода), например, к строковым данным, графике, файлам и другим.
- View System – богатый и расширяемый набор представлений (Views), который может быть использован для создания визуальных компонентов приложений, например, списков, текстовых полей, таблиц, кнопок или даже встроенного web-браузера.
- Location Manager – менеджер местоположения, позволяет приложениям периодически получать обновленные данные о текущем географическом положении устройства.
- Notification Manager – менеджер оповещений, благодаря которому все приложения могут отображать собственные уведомления для пользователя в строке состояния.

Таким образом, благодаря Application Framework, приложения в ОС Android могут получать в своё распоряжение вспомогательный функционал, благодаря чему реализуется принцип многократного использования компонентов приложений и операционной системы.

Естественно, в рамках политики безопасности.

Стоит отметить, просто на понятийном уровне, что фреймворк лишь выполняет код, написанный для него, в отличие от библиотек, которые исполняются сами.

Ещё одно отличие заключается в том, что фреймворк содержит в себе большое количество библиотек с разной функциональностью и назначением, в то время как библиотеки объединяют в себе наборы функций, близких по логике.

APPLICATIONS (ПРИЛОЖЕНИЯ)

На вершине программного стека Android лежит уровень приложений (Applications). Сюда относится набор базовых приложений, который предустановлен на ОС Android.

Например, в него входят браузер, почтовый клиент, программа для отправки SMS, карты, календарь, менеджер контактов и многие другие.

Список интегрированных приложений может меняться в зависимости от модели устройства и версии Android.

И помимо этого базового набора к уровню приложений относятся в принципе все приложения под платформу Android, в том числе и установленные пользователем.

Считается, что приложения под Android пишутся на языке Java, но нужно отметить, что существует возможность разрабатывать программы и на C/C++ (с помощью Native Development Kit), и на Basic (с помощью Simple) и с использованием других языков.

Также можно создавать собственные программы с помощью конструкторов приложений, таких как App Inventor.

Словом, возможностей тут много.

Архитектура операционной системы Андроид. Особенности, применение (продолжение)

Лекция 8

Внешнее хранилище в Android

Кроме перечисленных выше систем хранилищ данных, в Android есть и другой тип хранилища — так называемое внешнее хранилище (external storage — это название отражает изначальную задумку о том, что внешнее хранилище должно было располагаться на вставляемой в телефон внешней SD-карте).

По сути, внешнее хранилище играет роль домашней папки пользователя — именно там располагаются такие папки, как Documents, Download, Music и Pictures, именно внешнее хранилище открывают файловые менеджеры в качестве папки по умолчанию, именно к содержимому внешнего хранилища Android позволяет получить доступ компьютеру при подключении по кабелю.

```
# ls /sdcard
```

```
Alarms    Download    Podcasts
```

```
Android   Movies     Ringtones
```

```
Books     Music      Subtitles
```

```
DCIM     Notifications bluetooth
```

```
Documents Pictures
```

В отличие от внутреннего хранилища, разделённого на папки отдельных приложений, внешнее хранилище представляет собой «общую зону»: к нему есть полный доступ у любого приложения, получившего соответствующее разрешение от пользователя. Это разрешение стоит запрашивать таким приложениям, как файловый менеджер; а большинству остальных приложений лучше использовать intent с действием ACTION_GET_CONTENT, предоставляя пользователю возможность самому выбрать нужный файл в системном файловом менеджере.

Многие приложения предпочитают сохранять и некоторые из своих внутренних файлов, имеющие большой размер (например, кэш загруженных изображений и аудиофайлов) во внешнем хранилище.

Для этого Android выделяет приложениям во внешнем хранилище папки с названиями вида `Android/data/com.google.android.youtube`.

Самому приложению для доступа к такой папке не требуется разрешение на доступ ко всему внешнему хранилищу (поскольку в качестве владельца этой папки устанавливается его UID), но к этой папке может получить доступ любое другое приложение, имеющее такое разрешение, поэтому её, действительно, стоит использовать только для хранения публичных и некритичных данных.

При удалении приложения система удалит и его специальную папку во внешнем хранилище; но файлы, созданные приложениями во внешнем хранилище вне их специальной папки считаются принадлежащими пользователю и остаются на месте после удаления создавшего их приложения.

Исходно предполагалось, что внешнее хранилище действительно будет располагаться на внешней SD-карте, поскольку в то время объём SD-карт значительно превышал объём встраиваемой в телефоны памяти (в том же самом HTC Dream её было лишь 256 мегабайта, из которых на раздел data выделялось порядка 90 мегабайт).

С тех пор многие условия изменились; в современных телефонах часто нет слота для SD-карты, зато устанавливается огромное по мобильным меркам количество встроенной памяти (например, в Samsung Galaxy Note 9 её может быть до 512 гигабайт).

Поэтому в современном Android практически всегда и внутреннее, и внешнее хранилища располагаются во встроенной памяти.

Настоящий путь, по которому располагается внешнее хранилище в файловой системе, имеет форму /data/media/0 (для каждого пользователя устройства создаётся отдельное внешнее хранилище, и число в пути соответствует номеру пользователя). В целях совместимости до внешнего хранилища также можно добраться по путям /sdcard, /mnt/sdcard, /storage/self/primary, /storage/emulated/0, некоторым путям, начинающимся с /mnt/runtime/, и некоторым другим.

С другой стороны, у многих устройств всё-таки есть слот для SD-карты. Вставленную в Android-устройство SD-карту можно использовать как обычный внешний диск (не превращая её во внутреннее или внешнее хранилище системы) — сохранять на неё файлы, открывать хранящиеся на ней файлы, использовать её для перенесения файлов на другие устройства и т.п.

Кроме того, Android позволяет «заимствовать» SD-карту и разместить внутреннее и внешнее хранилище на ней (это называется заимствованным хранилищем — *adopted storage*).

При этом система переформатирует SD-карту и шифрует её содержимое — хранящиеся на ней данные невозможno прочесть, подключив её к другому устройству.

WEB или рабочий стол

Если задуматься об отличиях современных веб-приложений от «обычных» десктопных приложений, можно — среди недостатков — выделить несколько преимуществ веба:

- Веб-приложения переносимы между архитектурами и платформами, как и Java.
- Веб-приложения не требуют установки и всегда обновлены, как и Android Instant Apps.

Кроме того, веб-приложения существуют в виде страниц, которые могут ссылаться друг на друга — как в рамках одного сайта, так и между сайтами.

При этом страница на одном сайте не обязана ограничиваться ссылкой только на главную страницу другого, она может ссылаться на конкретную страницу внутри другого сайта (это называется *deep linking*).

Ссылаясь друг на друга, отдельные сайты объединяются в общую сеть, веб.

Несколько копий одной страницы — например, несколько профилей в социальной сети — могут быть одновременно открыты в нескольких вкладках браузера.

Интерфейс браузера рассчитан на переключение между одновременными сессиями (вкладками), а не между отдельными сайтами — в рамках одной вкладки вы можете перемещаться по ссылкам (и вперёд-назад по истории) между разными страницами разных сайтов.

Всё это противопоставляется «десктопу», где каждое приложение работает отдельно и часто независимо от других — и в этом плане то, как устроены приложения в Android, гораздо ближе к вебу, чем к «традиционным» приложениям

Деятельность и намерения

Основной вид компонентов приложений под Android — это activity.

Activity — это один «экран» приложения.

Activity можно сравнить со страницей в вебе и с окном приложения в традиционном оконном интерфейсе.

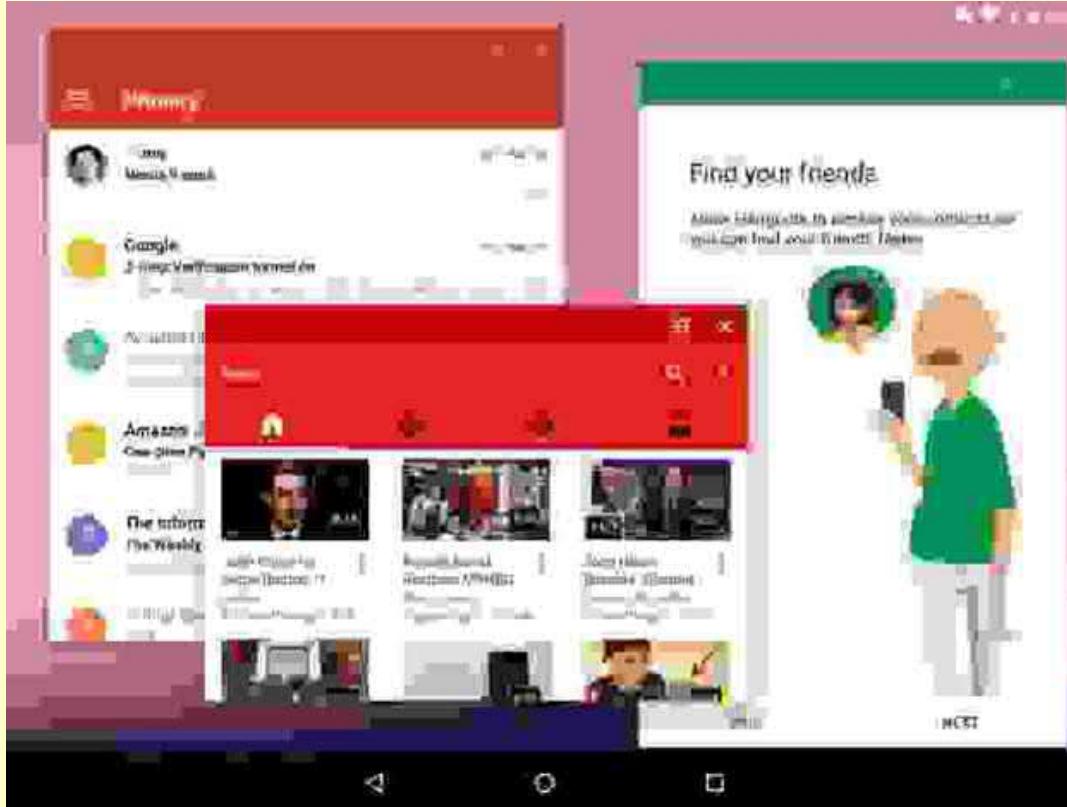
Окна

Собственно окна в Android тоже есть на более низком уровне — уровне window manager.

Каждой activity обычно соответствует своё окно.

Чаще всего окна activity развернуты на весь доступный экран, но:

- Во-первых, Android поддерживает мультиоконный режим — split-screen, picture-in-picture и даже freeform.
- Во-вторых, Android поддерживает подключение нескольких дисплеев.
- В-третьих, activity может намеренно занимать небольшую часть экрана (Theme_Dialog).



Например, в приложении для электронной почты (email client) могут быть такие activity, как Inbox Activity (список входящих писем), Email Activity (чтение одного письма), Compose Activity (написание письма) и Settings Activity (настройки).

Как и страницы одного сайта, activity одного приложения могут запускаться как друг из друга, так и независимо друг от друга (другими приложениями). Если в вебе на другую страницу обращаются по URL (ссылке), то в Android activity запускаются через intent'ы.

Intent — это сообщение, которое указывает системе, что нужно «сделать» (например, открыть данный URL, написать письмо на данный адрес, позвонить на данный номер телефона или сделать фотографию).

Приложение может создать такой intent и передать его системе, а система решает, какая activity (или другой компонент) будет его выполнять (handle).

Эта activity запускается системой (в существующем процессе приложения или в новом, если он ещё не запущен), ей передаётся этот intent, и она его выполняет.

Стандартный способ создавать intent'ы — через соответствующий класс в Android Framework.

Для работы с activity и intent'ами из командной строки в Android есть команда am — обёртка над стандартным классом Activity Manager:

```
# передаём -a ACTION -d DATA
```

```
# открыть сайт
```

```
$ am start -a android.intent.action.VIEW -d http://example.com
```

```
# позвонить по телефону
```

```
$ am start -a android.intent.action.CALL -d tel:+7-916-271-05-83
```

Intent'ы могут быть явными (explicit) и неявными (implicit).

Явный intent указывает идентификатор конкретного компонента, который нужно запустить — чаще всего это используется, чтобы запустить из одной activity другую внутри одного приложения (при этом intent может даже не содержать другой полезной информации).

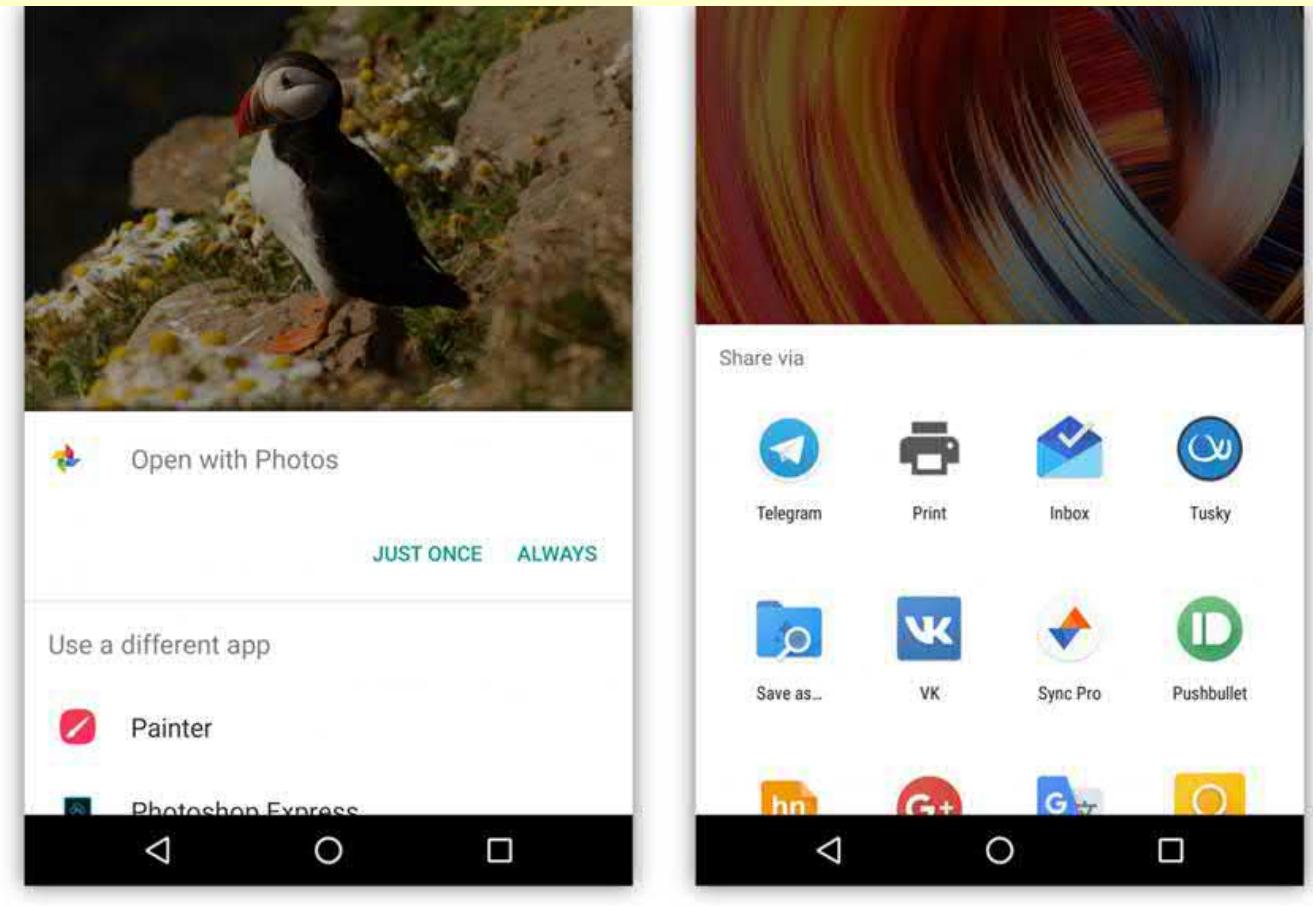
Неявный intent обязательно должен указывать действие, которое нужно сделать. Каждая activity (и другие компоненты) указывают в манифесте приложения, какие intent'ы они готовы обрабатывать (например, ACTION_VIEW для ссылок с доменом <https://example.com>).

Система выбирает подходящий компонент среди установленных и запускает его.

Если в системе есть несколько activity, которые готовы обработать intent, пользователю будет предоставлен выбор.

Обычно это случается, когда установлено несколько аналогичных приложений, например несколько браузеров или редакторов.

Кроме того, приложение может явно попросить систему показать диалог выбора (на самом деле при этом переданный intent оборачивается в новый intent с ACTION_CHOOSER) — это обычно используется для создания красивого диалога Share:



Кроме того, activity может вернуть результат в вызвавшую её activity.

Например, activity в приложении-камере, которая умеет обрабатывать intent «сделать фотографию» (ACTION_IMAGE_CAPTURE) возвращает сделанную фотографию в ту activity, которая создала этот intent.

При этом приложению, содержащему исходную activity, не нужно разрешение на доступ к камере.

Таким образом, правильный способ приложению под Android сделать фотографию — это не потребовать разрешения на доступ к камере и использовать Camera API, а создать нужный intent и позволить системному приложению-камере сделать фото.

Аналогично, вместо использования разрешения READ_EXTERNAL_STORAGE и прямого доступа к файлам пользователя стоит дать пользователю возможность выбрать файл в системном файловом менеджере (тогда исходному приложению будет разрешён доступ именно к этому файлу).

При этом «системное» приложение — не обязательно то, которое было предустановлено производителем (или автором сборки Android).

Все установленные приложения, которые умеют обрабатывать данный intent, в этом смысле равны между собой.

Пользователь может выбрать любое из них в качестве приложения по умолчанию для таких intent'ов, а может выбирать нужное каждый раз.

Выбранное приложение становится «системным» в том смысле, что пользователь выбрал, чтобы именно оно выполняло все задачи (то есть intent'ы) такого типа, возникающие в системе.

Лончер

Этой логике подчиняются даже такие «части системы», как, например, домашний экран (лончер, launcher).

Лончер — это специальное приложение со своими activity (которые используют специальные флаги вроде excludeFromRecents и launchMode="singleTask").

Нажатие кнопки «домой» создаёт intent категории HOME, который дальше проходит через обычный механизм обработки intent'ов — в том числе, если в системе установлено несколько лончёров и ни один не выбран в качестве лончера по умолчанию, система отобразит диалог выбора.

«Запуск» приложения из лончера тоже происходит через intent.

Лончер создаёт явный intent категории LAUNCHER, который «обрабатывается» запуском основной activity приложения.

Приложение может иметь несколько activity, которые поддерживают такой intent, и отображаться в лончере несколько раз (при этом может понадобиться указать им разную taskAffinity).

Или не иметь ни одной и не отображаться в лончере вообще (но по-прежнему отображаться в полном списке установленных приложений в настройках). «Обычные» приложения так делают довольно редко; самый известный пример такого поведения — Google Play Services.

Многие операционные системы делятся на собственно операционную систему и приложения, установленные поверх, ничего друг о друге не знающие и не умеющие взаимодействовать.

Система компонентов и intent'ов Android позволяет приложениям, по-прежнему абсолютно ничего друг о друге не зная, составлять для пользователя один интегрированный системный user experience — установленные приложения реализуют части одной большой системы, они составляют из себя систему.

И это, с одной стороны, происходит прозрачно для пользователя, с другой — представляет неограниченные возможности для кастомизации.

Задачи и задний стек

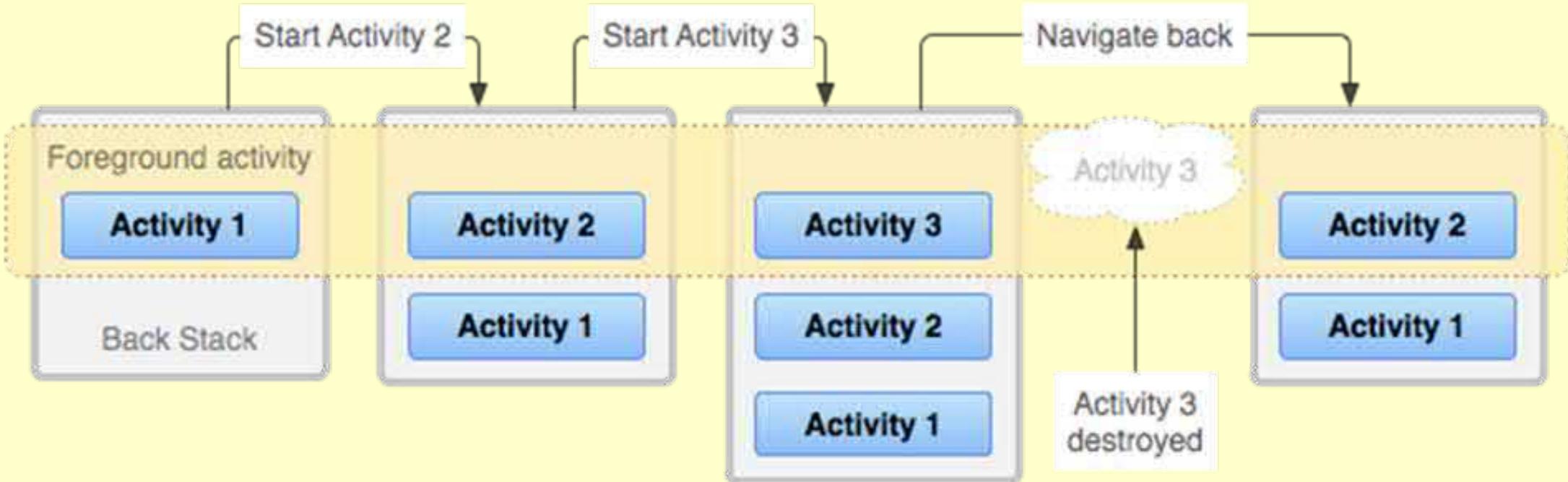
Как я уже говорил, в браузере пользователь может переключаться не между сайтами, а между вкладками, история каждой из которых может содержать много страниц разных сайтов.

Аналогично, в Android пользователь может переключаться между задачами (tasks), которые отображаются в виде карточек на recents screen.

Каждая задача представляет собой back stack — несколько activity, «наложенных» друг на друга.

Когда одна activity запускает другую, новая activity помещается в стек поверх старой.

Когда верхняя activity в стеке завершается — например, когда пользователь нажимает системную кнопку «назад» — предыдущая activity в стеке снова отображается на экране.



Каждый стек может включать в себя activity из разных приложений, и несколько копий одной activity могут быть одновременно открыты в рамках разных задач или даже внутри одного стека

При запуске новой activity могут быть указаны специальные флаги, такие как singleTop, singleTask, singleInstance и CLEAR_TOP, которые модифицируют этот механизм.

Например, приложения-браузеры обычно разрешают запуск только одной копии своей основной activity, и для переключения между открытыми страницами реализуют собственную систему вкладок.

С другой стороны, Custom Tabs — пример activity в браузере (чаще всего Chrome), которая ведёт себя «как обычно», то есть показывает только одну страницу, но позволяет одновременно открывать несколько своих копий.

Жизненный цикл приложения

Использование памяти.

Одно из основных ограничений встраиваемых и мобильных устройств — небольшое количество оперативной памяти (RAM).

Если современные устройства уже оснащаются несколькими десятками и сотнями гигабайт оперативной памяти, то в первом смартфоне на Android, HTC Dream (он же T-Mobile G1), вышедшем в сентябре 2008 года, её было всего 192 мегабайта.

Проблема ограниченной памяти дополнительно осложняется тем, что в мобильных устройствах, в отличие от «обычных» компьютеров, не используются swap-разделы (и swap-файлы) — в том числе и из-за низкой (по сравнению с SSD и HDD) скорости доступа к SD-картам и встроенной флеш-памяти, где они могли бы размещаться. Начиная с версии 4.4 KitKat, Android использует zRAM swap, то есть эффективно сжимает малоиспользуемые участки памяти. Тем не менее, проблема ограниченной памяти остаётся.

Если все процессы представляют собой для системы чёрный ящик, лучшая из возможных стратегия поведения в случае нехватки свободной памяти — принудительно завершать («убивать») какие-то процессы, что и делает Linux Out Of Memory (OOM) Killer.

Но Android знает, что происходит в системе, ему известно, какие приложения и какие их компоненты запущены, что позволяет реализовать гораздо более «умную» схему освобождения памяти.

Во-первых, когда свободная память заканчивается, Android явно просит приложения освободить ненужную память (например, сбросить кэш), вызывая методы `onTrimMemory`/`onLowMemory`.

Во-вторых, Android может эффективно проводить сборку мусора в фоновых приложениях, освобождая память, которую они больше не используют (на уровне Java), при этом не замедляя работу текущего приложения.

Но основной механизм освобождения памяти в Android — это завершение наименее используемых приложений.

Система автоматически выбирает приложения, наименее важные для пользователя (например, те, из которых пользователь давно ушёл), даёт их компонентам шанс дополнительно освободить ресурсы, вызывая такие методы, как `onDestroy`, и завершает их, полностью освобождая используемую ими память и ресурсы.

Если пользователь возвращается в `activity` приложения, завершённого системой из-за нехватки памяти, эта `activity` запускается снова.

При этом перезапуск происходит прозрачно для пользователя, поскольку `activity` сохраняет своё состояние при завершении (`onSaveInstanceState`) и восстанавливает его при последующем запуске.

Реализованные в Android Framework виджеты используют этот механизм, чтобы автоматически сохранить состояние интерфейса (UI) при перезапуске — с точностью до введённого в `EditText` текста, положения курсора, позиции прокрутки (`scroll`) и т.д.

Разработчик приложения может дополнительно реализовать сохранение и восстановление каких-то ещё данных, специфичных для этого приложения.

Сервисы

Приложениям может потребоваться выполнять действия, не связанные напрямую ни с какой activity, в том числе, продолжать делать их в фоне, когда все activity этого приложения завершены.

Например, приложение может скачивать из сети большой файл, обрабатывать фотографии, воспроизводить музыку, синхронизировать данные или просто поддерживать TCP-соединение с сервером для получения уведомлений.

Такую функциональность нельзя реализовывать, просто запуская отдельный поток — это было бы для системы чёрным ящиком; в том числе, процесс был бы завершён при завершении всех activity, независимо от состояния таких фоновых операций.

Вместо этого Android предлагает использовать ещё один вид компонентов — сервис.

Сервис нужен, чтобы сообщить системе, что в процессе приложения выполняются действия, которые не являются частью activity этого приложения.

Сам по себе сервис не означает создание отдельного потока или процесса — его точки входа (entry points) запускаются в основном потоке приложения.

Обычно реализация сервиса запускает дополнительные потоки и управляет ими самостоятельно.

Сервисы во многом похожи на activity: они тоже запускаются с помощью intent'ов и могут быть завершены системой при нехватке памяти.

Запущенные сервисы могут быть в трёх состояниях:

- Обслуживание на переднем плане (Foreground service) — сервис, выполняющий действие, состояние которого важно для пользователя, например, загрузка файла или воспроизведение музыки.

Такой сервис обязан отображать уведомление в системной шторке уведомлений (примеры: состояние загрузки, название текущей песни и управление воспроизведением).

Система считает такой сервис примерно настолько же важным для пользователя, как и текущая activity, и завершит его только в крайнем случае.

- Обслуживание на заднем плане (Background service) — сервис, выполняющий фоновое действие, состояние которого не интересует пользователя (чаще всего, синхронизацию).

Такие сервисы могут быть завершены при нехватке памяти с гораздо большей вероятностью.

В старых версиях Android большое количество одновременно запущенных фоновых сервисов часто становилось причиной «тормозов»; начиная с версии 8.0 Oreo, Android серьёзно ограничивает использование фоновых сервисов, принудительно завершая их через несколько минут после того, как пользователь выходит из приложения.

- Связанный сервис(Bound service) — сервис, обрабатывающий входящее Binder-подключение.

Такие сервисы предоставляют какую-то функциональность для других приложений или системы (например, WallpaperService и Google Play Services).

В этом случае система может автоматически запускать сервис при подключении к нему клиентов и останавливать его при их отключении.

Рекомендуемый способ выполнять фоновые действия — использование JobScheduler, системного механизма планирования фоновой работы. JobScheduler позволяет приложению указать критерии запуска сервиса, такие как:

- Доступность сети. Здесь приложение может указать, требуется ли этому сервису наличие сетевого подключения, и если да, то возможна ли работа в роуминге или при использовании лимитного (metered) подключения.
- Подключение к источнику питания, что позволяет сервисам выполняться, не «сажая батарею».
- Бездействие (idle), что позволяет сервисам выполнятся, пока устройство не используется, не замедляя работу во время активного использования.
- Обновления контента — например, появление новой фотографии.
- Период и крайний срок запуска — например, очистка кэша может производиться ежедневно, а синхронизация событий в календаре — каждый час.

JobScheduler планирует выполнение (реализованное как вызов через Binder) зарегистрированных в нём сервисов в соответствии с указанными критериями.

Поскольку JobScheduler — общесистемный механизм, он учитывает при планировке критерии зарегистрированных сервисов всех установленных приложений.

Например, он может запускать сервисы по очереди, а не одновременно, чтобы предотвратить резкую нагрузку на устройство во время использования, и планировать периодическое выполнение нескольких сервисов небольшими группами (batch), чтобы предотвратить постоянное энергозатратное включение-выключение радиооборудования.

TCP-соединение

Как можно заметить, использование JobScheduler не может заменить собой одного из вариантов использования фоновых сервисов — поддержания TCP-соединения с сервером для получения push-уведомлений.

Если бы Android предоставлял приложениям такую возможность, устройству пришлось бы держать все приложения, соединяющиеся со своими серверами, запущенными всё время, а это, конечно, невозможно.

Решение этой проблемы — специальные push-сервисы, самый известный из которых — Firebase Cloud Messaging от Google (бывший Google Cloud Messaging).

Клиентская часть FCM реализована в приложении Google Play Services.

Это приложение, которое специальным образом исключается из обычных ограничений на фоновые сервисы, поддерживает одно соединение с серверами Google.

Разработчик, желающий отправить своему приложению push-уведомление, пересыпает его через серверную часть FCM, после чего приложение Play Services, получив сообщение, передаёт его приложению, которому оно предназначено.

Такая схема позволяет, с одной стороны, мгновенно доставлять push-уведомления всем приложениям (не дожидаясь следующего периода синхронизации), с другой стороны, не держать множество приложений одновременно запущенными.

Приемники вещания и поставщики контента

Кроме activity и сервисов, у приложений под Android есть два других вида компонентов, менее интересных для обсуждения — это broadcast receiver'ы и content provider'ы.

Broadcast receiver — компонент, позволяющий приложению принимать 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 посыпает 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.

Content provider — компонент, позволяющий приложению предоставлять другим приложениям доступ к данным, которыми оно управляет. Пример данных, доступ к которым можно получить таким образом — список контактов пользователя.

При этом приложение может хранить сами данные каким угодно образом, в том числе на устройстве в виде файлов, в настоящей базе данных (SQLite) или запрашивать их с сервера по сети.

В этом смысле content provider — это унифицированный интерфейс для доступа к данным, независимо от формы их хранения.

Взаимодействие с content provider'ом во многом похоже на доступ к удалённой базе данных через REST API.

Приложение-клиент запрашивает данные по URI (например, content://com.example.Dictionary.provider/words/42) через ContentResolver.

Приложение-сервер определяет, к какому именно набору данных был сделан запрос, используя UriMatcher, и выполняет запрошенное действие (query, insert, update, delete).

Именно поверх content provider'ов реализован Storage Access Framework, позволяющий приложениям, хранящим файлы в облаке (например, Dropbox и Google Photos) предоставлять доступ к ним остальным приложениям, не занимая место на устройстве полной копией всех хранящихся в облаке файлов.

Загрузка

Традиционный подход к безопасности компьютерных систем ограничивается тем, чтобы защищать систему от программных атак.

Считается, что если у злоумышленника есть физический доступ к компьютеру, игра уже проиграна: он может получить полный доступ к любым хранящимся на нём данным.

Для этого ему достаточно, например, запустить на этом компьютере произвольную контролируемую им операционную систему, позволяющую ему обойти любые накладываемые «основной» системой ограничения прав, или напрямую подключить диск с данными к другому устройству.

При желании, злоумышленник может оставить компьютер в работоспособном состоянии, но пропатчить установленную на нём систему, установить произвольные бэкдоры, кейлоггеры и т.п.

Именно на защиту от программных атак ориентирована модель ограничения прав пользователей в Unix (и основанная на ней технология app sandbox в Android); само по себе ограничение прав в Unix никак не защищает систему от пользователя, пробравшегося в серверную и получившего физический доступ к компьютеру.

И если серьёзные многопользовательские сервера можно и нужно охранять от неавторизованного физического доступа, к персональным компьютерам — а тем более мобильным устройствам — такой подход просто неприменим.

Пытаться улучшать ситуацию с защитой от злоумышленника, получившего физический доступ к устройству, можно по двум направлениям:

- Во-первых, можно шифровать хранящиеся на диске данные, тем самым не позволяя злоумышленнику получить доступ к самим данным, даже если у него есть доступ к содержимому диска.
- Во-вторых, можно так или иначе ограничивать возможность загрузки на устройстве произвольных операционных систем, вынуждая злоумышленника проходить процедуры аутентификации и авторизации в установленной системе.

Именно с этими двумя направлениями защиты связана модель безопасной загрузки в Android.

Проверенная загрузка (Verified Boot)

Процесс загрузки Android построен так, что он, с одной стороны, не позволяет злоумышленникам загружать на устройстве произвольную ОС, с другой стороны, может позволять пользователям устанавливать кастомизированные сборки Android (и другие системы).

Прежде всего, Android-устройства, в отличие от «десктопных» компьютеров, обычно не позволяют пользователю (или злоумышленнику) произвести загрузку со внешнего носителя; вместо этого сразу запускается установленный на устройстве bootloader (загрузчик). Bootloader — это относительно простая программа, в задачи которой (при загрузке в обычном режиме) входят:

- инициализация и настройка Trusted Execution Environment (например, ARM TrustZone),
- нахождение разделов встроенной памяти, в которых хранятся образы ядра Linux и initramfs,
- проверка их целостности и неприкосновенности (integrity) — в противном случае загрузка прерывается с сообщением об ошибке — путём верификации цифровой подписи производителя,
- загрузка ядра и initramfs в память и передача управления ядру.

Flashing, unlocking, fastboot и recovery

Кроме того, bootloader поддерживает дополнительную функциональность для обновления и переустановки системы.

Во-первых, это возможность загрузить вместо основной системы (Android) специальную минимальную систему, называемую *recovery*.

Версия *recovery*, устанавливаемая на большинство Android-устройств по умолчанию, очень минималистична и поддерживает только установку обновлений системы в автоматическом режиме, но многие энтузиасты Android устанавливают кастомную *recovery*.

Это делается путём использования второй «фичи» bootloader'a, направленной на обновление и переустановку системы — поддержки перезаписи (*flashing*) содержимого и структуры разделов по командам с подсоединённого по кабелю компьютера.

Для этого bootloader способен загружаться в ещё один специальный режим, который называют *fastboot mode* (или иногда просто *bootloader mode*), поскольку обычно для общения между компьютером и bootloader'ом в этом режиме используется протокол *fastboot* (и соответствующий ему инструмент *fastboot* из Android SDK со стороны компьютера).

Некоторые реализации bootloader'a используют другие протоколы.

В основном это касается устройств, выпускаемых компанией Samsung, где специальная реализация bootloader'a (Loke) общается с компьютером по собственному проприетарному протоколу (Odin).

Для работы с Odin со стороны компьютера можно использовать либо реализацию от самих Samsung (которая тоже называется Odin), либо свободную реализацию под названием Heimdall.

Конкретные детали зависят от реализации bootloader'a (то есть различаются в зависимости от производителя устройства), но во многих случаях установка recovery и сборок Android, подписанных ключом производителя устройства, просто работает без дополнительных сложностей:

```
$ fastboot flash recovery recovery.img
```

```
$ fastboot flash boot boot.img
```

```
$ fastboot flash system system.img
```

Таким образом можно вручную обновлять систему; а вот установить более старую версию не получится: функция, известная как защита от отката, не позволит bootloader'у загрузить более старую версию Android, чем была загружена в прошлый раз, даже если она подписана ключом производителя, поскольку загрузка старых версий открывает дорогу к использованию опубликованных уязвимостей, которые исправлены в более новых версиях.

Кроме того, многие устройства поддерживают разблокировку bootloader'a (unlocking the bootloader, также известную как OEM unlock) — отключение проверки bootloader'ом подписи системы и recovery, что позволяет устанавливать произвольные сборки того и другого (у части производителей это аннулирует гарантию).

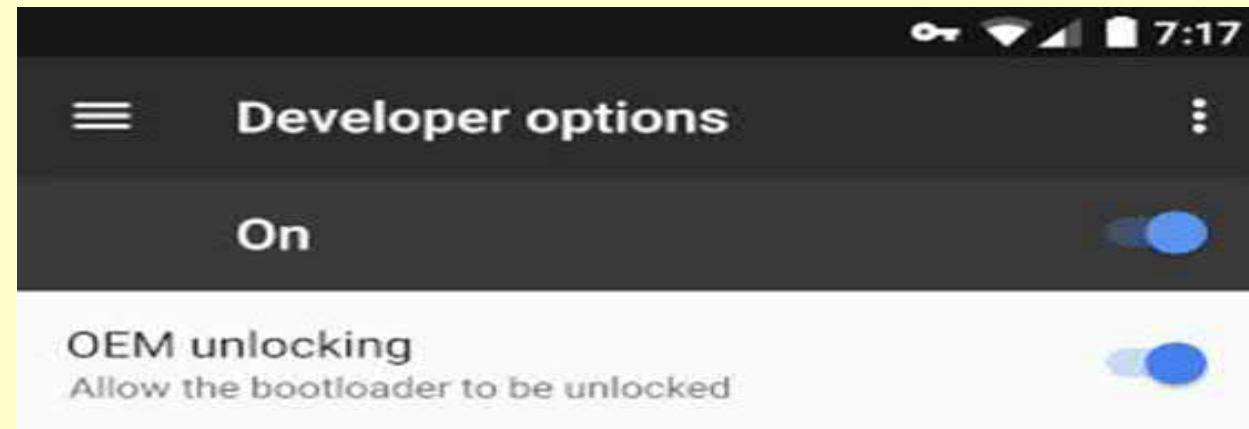
Именно так обычно устанавливаются такие популярные дистрибутивы Android, как LineageOS (бывший CyanogenMod), Paranoid Android, AOKP, OmniROM и другие.

Поскольку разблокировка bootloader'a всё-таки позволяет загрузить на устройстве собственную версию системы, в целях безопасности при разблокировке все пользовательские данные (с раздела data) принудительно удаляются.

Если систему переустанавливает сам пользователь, а не злоумышленник, после пере установки он может восстановить свои данные из бэкапа (например, из облачного бэкапа на серверах Google или из бэкапа на внешнем носителе), если злоумышленник — он получит работающую систему, но не сможет украсть данные владельца устройства.

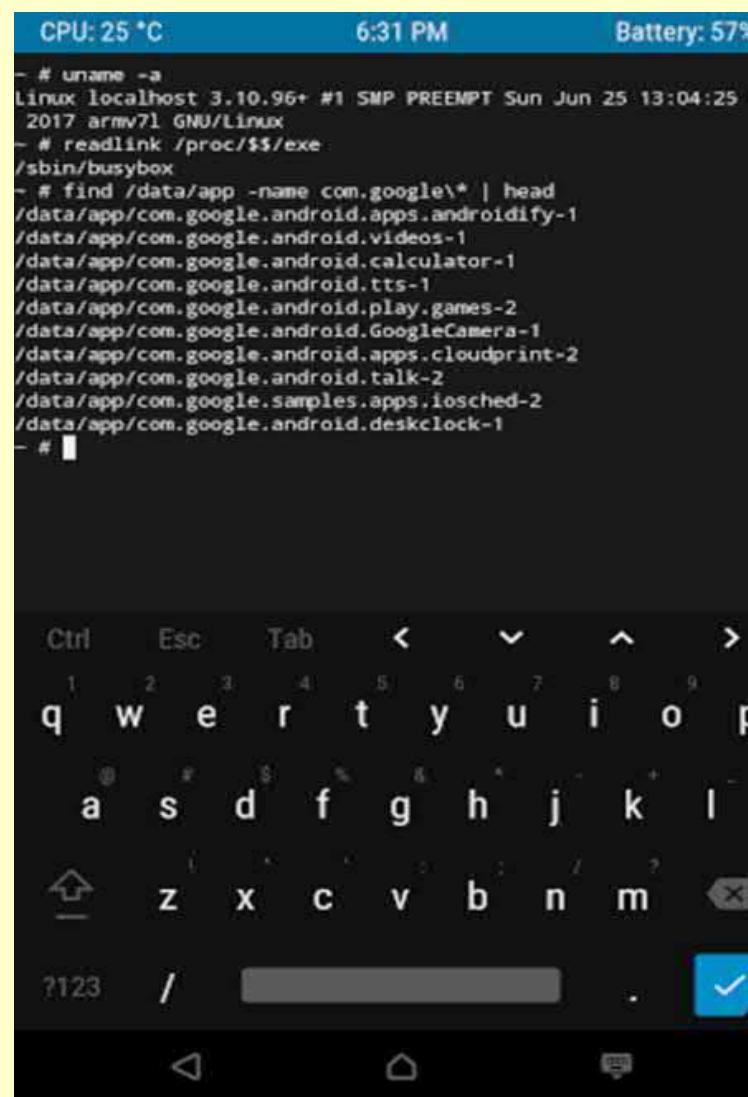
После установки предпочтаемых сборок recovery и системы bootloader стоит заблокировать обратно, чтобы снова защитить свои данные в случае попадания устройства в руки злоумышленников.

Для разблокировки bootloader'a может также потребоваться дополнительно разрешить её из настроек системы:



Популярная сторонняя recovery, которую устанавливают большинство «флешаголиков» (flashaholics) — TWRP (Team Win Recovery Project).

Она содержит тач-интерфейс и множество продвинутых «фич», в том числе возможность устанавливать части системы из сборок в виде zip-архивов, встроенную поддержку бэкапов и даже полноценный эмулятор терминала с виртуальной клавиатурой:



Шифрование диска

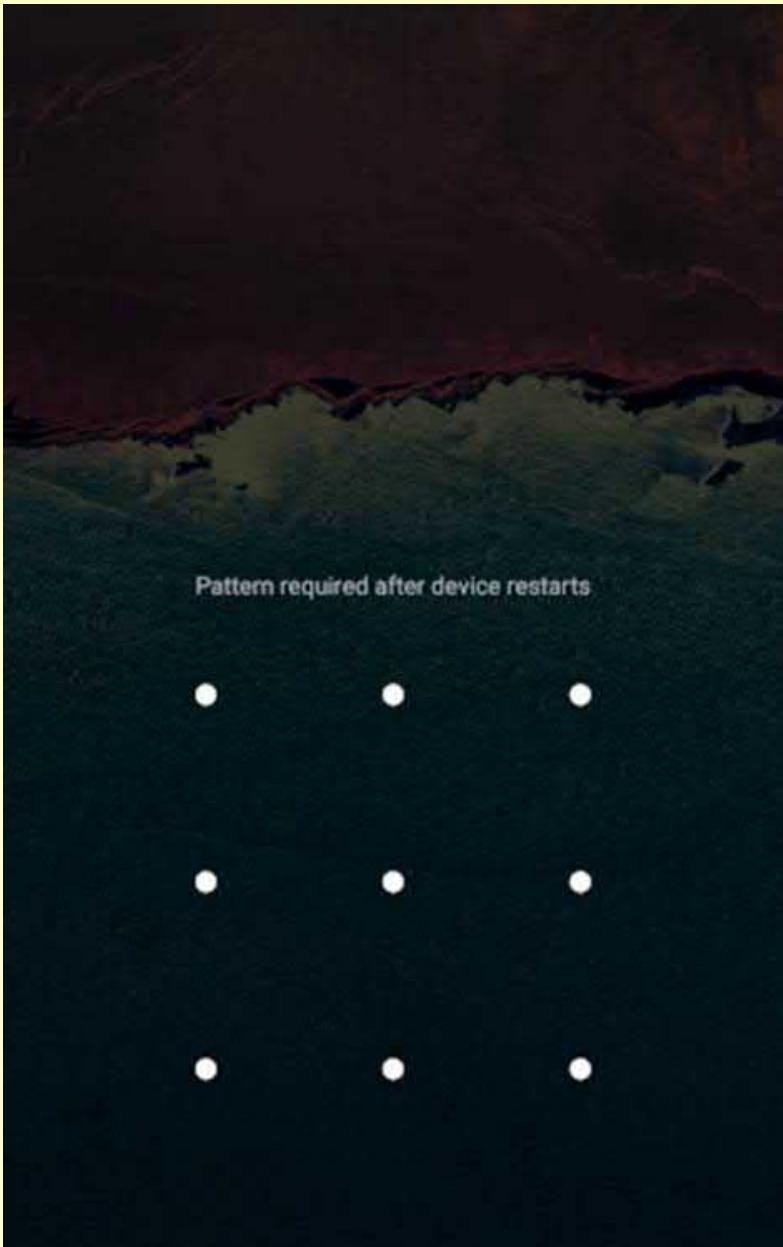
Современные версии Android используют пофайловое шифрование данных (file-based encryption).

Этот механизм основан на встроенной в ext4 поддержке шифрования, реализованной в ядре Linux (fscrypt), и позволяет системе зашифровывать различные части файловой системы различными ключами.

По умолчанию система шифрует большинство данных пользователя, расположенных на разделе data, с помощью ключа, который создаётся на основе пароля пользователя и не сохраняется на диск (credential encrypted storage).

Это означает, что при загрузке система должна попросить пользователя ввести свой пароль, чтобы вычислить с его помощью ключ для расшифровки данных.

Именно поэтому первый раз после включения устройства пользователя встречает требование ввести полный пароль или графический ключ, а не просто пройти аутентификацию, приложив палец к сканеру отпечатков.



Pattern required after device restarts

В дополнение к credential encrypted storage в Android также используется device encrypted storage — шифрование ключом на основе данных, которые хранятся на устройстве (в том числе в Trusted Execution Environment).

Файлы, зашифрованные таким образом, система может расшифровать до того, как пользователь введёт пароль.

Это лежит в основе функции, известной как Direct Boot: система способна загружаться в некоторое работоспособное состояние и без ввода пароля; при этом приложения могут явно попросить систему сохранить (наименее приватную) часть своих данных в device encrypted storage, что позволяет им начинать выполнять свои базовые функции, не дожидаясь полной разблокировки устройства.

Например, Direct Boot позволяет будильнику срабатывать и до первого ввода пароля, что особенно полезно, если устройство непредвиденно перезагружается ночью из-за временного отключения питания или сбоя системы.

Root

Так называемый root-доступ — это возможность выполнять код от имени «пользователя root» (UID 0, также известного как суперпользователь).

Напомню, что root — это специально выделенный Unix-пользователь, которому — за несколькими интересными исключениями — разрешён полный доступ ко всему в системе, и на которого не распространяются никакие ограничения прав.

Как и большинство других современных операционных систем, Android спроектирован с расчётом на то, что обычному пользователю ни для чего не требуется использовать root-доступ.

В отличие от более закрытых операционных систем, пользователи которых называют разрушение наложенных на них ограничений буквально « побегом из тюрьмы », в Android прямо « из коробки », без необходимости получать root-доступ и устанавливать специальные сторонние « твики », есть возможность:

- устанавливать произвольные приложения — как из множества существующих магазинов приложений, так и произвольные APK из любых источников,
- выбирать приложения по умолчанию: веб-браузер, email-клиент, камеру, файловый менеджер, лончер, приложение для звонков, приложение для СМС, приложение для контактов и так далее (это работает за счёт красивой системы activity и intent'ов, которую я описал в прошлой статье),
- устанавливать и использовать пакеты значков (icon packs),
- получать прямой доступ к файловой системе, хранить в ней произвольные файлы,
- подключать устройство к компьютеру (или даже к другому Android-устройству) и напрямую передавать между ними файлы по кабелю,
- и даже, во многих дистрибутивах Android, настраивать цвета и шрифты системной темы.

Итак, для обычновенных задач, с которыми может столкнуться простой пользователь, root-доступ не нужен.

В то же время использование root-доступа неизбежно сопряжено со многими проблемами с безопасностью (подробнее о них ниже), поэтому в большинстве случаев Android не позволяет пользователю работать от имени root.

Приложения, в том числе эмуляторы терминалов, выполняются от имени своих ограниченных Unix-пользователей; а shell, который запускается при использовании команды adb shell, работает от имени специально для этого предназначенного Unix-пользователя shell.

Тем не менее, бывает, что root доступен пользователю:

- Во-первых, root-доступ обычно разрешён на сборках Android для эмуляторов — поставляемых вместе с Android Studio виртуальных машин на базе QEMU, которые изображают реальные устройства и обычно используются разработчиками для отладки и тестирования приложений.

- Во-вторых, root-доступ включён по умолчанию во многих сторонних дистрибутивах Android (pre-rooted ROMs).
- В-третьих, при разблокированном bootloader'e root-доступ на любой сборке Android можно включить напрямую, просто установив исполняемый файл, реализующий команду su, через bootloader.
- Ну и наконец, в-четвёртых, на старых версиях системы, содержащих известные уязвимости, часто можно получить root-доступ, проэксплуатировав их (обычно для получения root-доступа эксплуатируются сразу несколько уязвимостей в разных слоях и компонентах системы).

Например, если система не обновлялась с первой половины 2016 года, для получения root-доступа можно воспользоваться получившей широкую известность уязвимостью Dirty Cow.

Зачем это может быть нужно? Конечно, root-доступ полезен для отладки и исследования работы системы.

Кроме того, обладая root-доступом, можно неограниченно настраивать систему, изменяя её темы, поведение и многие другие аспекты.

Можно принудительно подменять код и ресурсы приложений — например, можно удалить из приложения рекламу или разблокировать его платную или скрытую функциональность.

Можно устанавливать, изменять и удалять произвольные файлы, в том числе в разделе *system* (хотя это почти наверняка плохая идея).

С более философской точки зрения, root-доступ позволяет пользователю полностью контролировать своё устройство и свою систему — вместо того, чтобы устройство и разработчики ПО контролировали пользователя.

С большими возможностями приходит и большая ответственность, и совсем не всегда эту ответственность можно доверить пользователю и приложениям.

Другими словами, с root-доступом связано большое количество проблем в области безопасности и стабильности системы.

Напомню, что пользователь почти всегда взаимодействует с устройством не напрямую, а через приложения.

А это значит, что и root-доступ он будет использовать в основном через приложения, которые, можно надеяться, будут добросовестно пользоваться root-доступом для хороших целей.

Но если на устройстве доступен root, это, в принципе, означает, что приложения могут воспользоваться им и для нехороших целей — навредить системе, украсть ценные данные, заразить систему вирусом, установить кейлоггер и т.п.

Как я уже говорил, в отличие от традиционной модели доверия программам в классическом Unix, Android рассчитан на то, что пользователь не может доверять сторонним приложениям — поэтому их и помещают в песочницу.

Тем более нельзя доверять приложениям root-доступ, надеясь, что они будут использовать его только во благо.

Фактически, root-доступ разрушает аккуратно выстроенную модель безопасности Android, снимая с приложений все ограничения и открывая им права на доступ ко всему в системе.

С другой стороны, и приложения не могут доверять устройству, на котором подключен root-доступ, поскольку на таком устройстве в его работу имеют возможность непредусмотренными способами вмешиваться пользователь и остальные приложения.

Например, разработчики приложения, содержащего платную функциональность, естественно, не захотят, чтобы проверку совершения покупки можно было отключить.

Многие приложения, работающие с особенно ценными данными — например, Google Pay (бывший Android Pay) — явно отказываются работать на устройствах с root-доступом, считая их недостаточно безопасными.

Google Play и root-доступ

Google разрешает производителям предустанавливать Google Play Store и Google Play Services только на устройства, где root-доступ отключён, тем самым делая Android более привлекательной платформой для разработчиков, которые, естественно, предпочитают вкладывать ресурсы в разработку под платформы, не позволяющие пользователям с лёгкостью вмешиваться в работу их приложений.

А поскольку Play Store — наиболее известный и популярный (как среди разработчиков приложений, так и среди пользователей) магазин приложений для Android, большинство производителей предпочитают предустанавливать его на свои устройства.

Есть и исключения — например, Amazon использует собственный дистрибутив Android под названием Fire OS для своих устройств — Echo, Fire TV, Fire Phone, Kindle Fire Tablet — и не предустанавливает никаких приложений от Google.

Именно поэтому root-доступ по умолчанию отключён на большинстве популярных Android-устройств.

Несмотря на ограничение для производителей, Google разрешает пользователям сборок, в которых разрешён root-доступ, самостоятельно устанавливать Google Play (обычно это делается путём установки готовых пакетов от проекта Open GApps).

При этом Google требует, чтобы после установки пользователь вручную зарегистрировал устройство на специально предназначеннной для этого странице регистрации.

Ограничение доступа к root

В «обычном» Linux, как и в других Unix-системах, для получения root-доступа (с помощью команд sudo и su) пользователю требуется ввести пароль (или свой, или пароль Unix-пользователя root, в зависимости от настройки системы и конкретной команды) или авторизоваться другим способом (например, с помощью отпечатка пальца).

В дополнение к собственно авторизации это служит подтверждением того, что пользователь доверяет этой программе и согласен предоставить ей возможность воспользоваться root-доступом

Стандартная версия команды su из Android Open Source Project не запрашивает подтверждения пользователя явно, но она доступна только Unix-пользователю shell (и самому root), что полностью отрезает приложениям возможность получать root-права.

Многие сторонние реализации su позволяют любому приложению получать права суперпользователя, что, как я объяснил выше, очень плохо в плане безопасности.

В качестве компромисса многими пользователями используются специальные программы, которые управляют доступом к su.

При попытке приложения вызвать su они запрашивают подтверждение у пользователя, который может разрешить или запретить приложению получить root-доступ.

Несколько лет назад была популярна одна из таких программ, SuperSU; в последнее время её вытеснил новый открытый проект под названием Magisk.

Magisk

Основная особенность Magisk — возможность проведения широкого класса модификаций системы, в том числе связанных с изменением файлов, расположенных в папке `/system`, на самом деле не изменяя сам раздел `system` (это называют словом `systemless-ly`), путём использования нескольких продвинутых «фич» Linux.

В сочетании с Magisk Hide — возможностью не просто не давать некоторым приложениям root-доступ, а полностью прятать от них сам факт наличия root-доступа и установки Magisk в системе — это позволяет устройству по-прежнему получать обновления системы от производителя и использовать приложения вроде того же Google Pay, которые отказываются работать на root-ованных системах.

Magisk Hide способен обходить даже такие достаточно продвинутые технологии обнаружения root-а, как SafetyNet от Google.

Хотя это действительно невероятно круто, нужно осознавать, что возможность использовать приложения вроде Google Pay на root-ованных устройствах несмотря на встроенные в них проверки — это не решение связанных с root-доступом проблем с безопасностью.

Приложения, которым пользователь доверил root-доступ, по-прежнему могут решить вмешаться в работу системы и украсть деньги пользователя через Google Pay.

Проблемы с безопасностью остаются, нам лишь удаётся закрыть на них глаза.

Magisk поддерживает systemless-ную установку готовых системных «твиков» в виде Magisk Modules, специальных пакетов для установки с использованием Magisk.

В таком виде доступны, например, известный root-фреймворк Xposed и ViPER, набор продвинутых драйверов и настроек для воспроизведения звука.

SoC, драйвера и фрагментация

Системы на кристалле

Аппаратное обеспечение традиционных настольных и серверных компьютеров построено вокруг материнской платы, на которую устанавливаются такие важнейшие компоненты «харда», как центральный процессор и оперативная память, и к которой затем подключаются дополнительные платы — например, графическая и сетевая — содержащие остальные компоненты (соответственно, графический процессор и Wi-Fi адаптер).

В отличие от такой схемы, в большинстве Android-устройств используются так называемые системы на кристалле (*system on a chip*, SoC).

Система на кристалле представляет собой набор компонентов компьютера — центральный процессор, блок оперативной памяти, порты ввода-вывода, графический процессор, LTE-, Bluetooth- и Wi-Fi-модемы и т.п. — полностью реализованных и интегрированных в рамках одного микрочипа.

Такой подход позволяет не только уменьшить физический размер устройства, чтобы оно поместилось в кармане, и повысить его производительность за счёт большей локальности и лучшей интеграции между компонентами, но и значительно снизить его энергопотребление и тепловыделение, что особенно актуально для мобильных устройств, питающихся от встроенной батарейки и не имеющих систем активного охлаждения.

Но, конечно, системы на кристалле имеют и свои недостатки.

Наиболее очевидный недостаток состоит в том, что такую систему нельзя «проапгрейдить», докупив, например, дополнительной оперативной памяти, как нельзя и заменить плохо работающий компонент.

Это, в принципе, выгодно компаниям-производителям, поскольку побуждает людей покупать новые устройства, когда существующие морально устаревают или выходят из строя, вместо того, чтобы их точечно обновлять или ремонтировать.

Драйвера

Другой — гораздо более важный именно в контексте Android — недостаток SoC заключается в связанных с ними сложностях с драйверами.

Как и в традиционных системах, основанных на отдельных платах, каждый компонент системы на кристалле — от камеры до LTE-модема — требует для работы использования специальных драйверов.

Но в отличие от традиционных систем, эти драйвера обычно разрабатываются производителями систем на кристалле и специфичны для их конкретных моделей; кроме того, исходные коды таких драйверов обычно не раскрываются, да и бинарные сборки в свободный доступ производителями выкладываться совсем не всегда.

Вместо этого производитель SoC (например, Qualcomm) отдаёт готовую сборку драйверов производителям Android-устройств (например, Sony или LG), которые включают её в свою сборку Android, основанную на коде из Android Open Source Project.

Так и получается, что сборка Android, предустановленная на устройстве производителем, содержит все нужные для этого устройства драйвера, а напрямую использовать сборку для одного устройства на другом невозможно.

В результате авторам сборок Android приходится прилагать отдельные усилия для поддержки каждого семейства моделей Android-устройств.

Сами производители устройств совсем не всегда заинтересованы в том, чтобы выделять ресурсы на портирование своих сборок на новые версии Android.

В то же время разработчикам сторонних дистрибутивов Android приходится извлекать драйвера из сборок, предустанавливаемых производителями, что связано с дополнительными сложностями и не всегда приводит к хорошему результату.

У многих сторонних дистрибутивов хватает ресурсов только для поддержки наиболее популярных моделей устройств

Фрагментация

Это приводит к проблеме, известной как фрагментация: экосистема Android состоит из большого количества различных сборок — как официальных сборок от производителей устройств, так и версий сторонних дистрибутивов.

Многие из них к тому же основаны на старых версиях Android, поскольку для многих устройств обновления сборок от производителя выходят медленно или не выходят совсем.

Конечно, медленное обновление экосистемы означает, что не все пользователи получают небольшие обновления интерфейса и другие видимые пользователю улучшения, которые приносят новые версии Android.

Но оно приносит и две гораздо более серьёзные проблемы.

Во-первых, страдает безопасность.

Хотя современные системы используют продвинутые механизмы защиты, в них постоянно обнаруживаются новые уязвимости.

Обычно эти уязвимости оперативно исправляются разработчиками, но это не помогает пользователям, если обновления системы, содержащие исправления, до них никогда не доходят.

Во-вторых, фрагментация отрицательно сказывается на разработчиках приложений под Android — страдает так называемый developer experience (DX, по аналогии с user experience/UX).

В теории, несмотря на внутренние различия в драйверах и пользовательском интерфейсе разных версий и сборок Android, используемые разработчиками приложений API — Android Framework, OpenGL/Vulkan и другие — должны быть переносимы и работать одинаково.

На практике это, конечно, не всегда так, и разработчикам приходится тестировать и обеспечивать работу своих приложений на множестве версий и сборок Android — на разных устройствах, разных версиях системы, сторонних дистрибутивах и так далее.

Далеко не все производители Android-устройств не считают важным своевременно выпускать обновления для своих устройств.

Например, Google выпускают обновления для своих линеек Nexus и Pixel одновременно с тем, как выходит новая версия Android.

У многих других производителей портирование сборок Android на его новую версию занимает месяцы, но они, тем не менее, стараются выпускать ежемесячные обновления безопасности.

Кроме того, совсем не все устройства, где используется Android, построены на основе SoC.

Android вполне можно установить и на обычный «десктопный» компьютер (подойдут, например, сборки от проектов Android-x86 и RemixOS).

Специальная сборка Android встроена в ChromeOS, что позволяет Chromebook'ам запускать Android-приложения наряду с Linux-приложениями и веб-приложениями.

Аналогичного подхода — запуска специальной сборки Android в контейнере — придерживается проект Anbox, позволяющий использовать Android-приложения на «обыкновенных» Linux-системах. (Напомню, что Android-приложения так легко переносятся на x86-архитектуры, не требуя перекомпиляции, благодаря использованию виртуальной машины Java,

Project Treble

Наиболее прямой способ бороться с фрагментацией — это всячески убеждать производителей устройств не забрасывать поддержку своих сборок Android.

Как показывает практика, это работает, но работает недостаточно хорошо. Было бы гораздо лучше, если можно было бы разработать техническое решение, повышающее переносимость сборок Android и тем самым серьёзно облегчающее задачу авторов сборок и сторонних дистрибутивов.

И такое решение уже существует. В 2017 году Google анонсировали Project Treble — новую, ещё более модульную (по сравнению с уже существующим HAL, hardware abstraction layer) архитектуру взаимодействия драйверов (и остального софта, специфичного для конкретного устройства) с остальными частями системы.

Treble позволяет устанавливать основную систему и драйвера, специфичные для устройства, на разные разделы файловой системы, и обновлять — или как угодно изменять — систему отдельно от установленных драйверов.

Treble в корне меняет ситуацию с медленным выпуском обновлений и плохой переносимостью сборок.

Благодаря Treble устройства от семи разных компаний (Sony, Nokia, OnePlus, Oppo, Xiaomi, Essential и Vivo — а не только от самих Google) смогли участвовать в бета-программе Android Pie.

Treble позволил Essential выпустить обновление до Android Pie для своего Essential Phone прямо в день выхода Android Pie.

Одну и ту же сборку Android — один и тот же бинарный файл без перекомпиляции или каких-либо изменений — теперь можно запускать на любом устройстве, поддерживающем Treble, несмотря на то, что они могут быть основаны на совершенно разных SoC.

Влияние Treble действительно сложно переоценить.

Java принесла возможность «write once, run everywhere» для высокоуровневого кода — в том числе и возможность запускать Android-приложения на компьютерах с практически любой архитектурой процессора. Treble — аналогичный прорыв, позволяющий использовать однажды написанную и скомпилированную сборку Android на устройствах с совершенно разными SoC.

Теперь дело за производителями, которым нужно конвертировать свои драйвера в формат, совместимый с Treble.

Можно надеяться, что через несколько лет проблемы с обновлениями Android-устройств исчезнут окончательно.

Механизм(компонента) разделяемой памяти(Ashmem)

Anonymous Shared Memory (ashmem) — механизм разделяемой памяти. В Линуксе, как правило, данный механизм реализован через POSIX SHM. Разработчики Андроида сочли его недостаточно защищённым, что могло сыграть на руку вредоносному ПО.

Особенностями ashmem-а являются счётчик ссылок, при обнулении которого разделяемая память может быть освобождена (например, память освобождается при завершении всех процессов, использующих её), и сокращение разделяемого региона при нехватке памяти в системе.

Ярким примером использования ashmem-а является процесс zygote, в котором загружается стартовая версия Dalvik VM (или ART) с загруженными базовыми классами и ресурсами, а остальные приложения просто ссылаются на эту память.

Binder имеет ограничение на размер транзакции в 1МБ (иначе будет выброшено исключение TransactionTooLargeException).

Если нам надо передать из одного процесса в другой большой объём данных, можно как раз воспользоваться Ashmem-ом: создать MemoryFile и передать дескриптор файла в другой процесс.

Logger

Обычные дистрибутивы, как правило, используют две системы логирования: лог ядра, доступный через команду `dmesg`, и системные логи, располагающиеся обычно в директории `/var/log`.

Система Андроида включает несколько циклических буферов для хранения сообщений пользовательских программ (что продлевает время жизни карт памяти, так как циклы чтения-записи не расходуются впустую) и не имеет дополнительных задержек от работы с сокетами, которые применяются в стандартном `syslog`-е.

На диаграмме представлена общая система логирования Андроида. Драйвер логирования предоставляет доступ к каждому буферу через `/dev/log/*`. Приложения имеют доступ к ним не напрямую, а через библиотеку `liblog`. С библиотекой `liblog` общаются классы `Log`, `Slog` и `EventLog`. Команда `adb logcat` показывает содержимое буфера “`main`”.

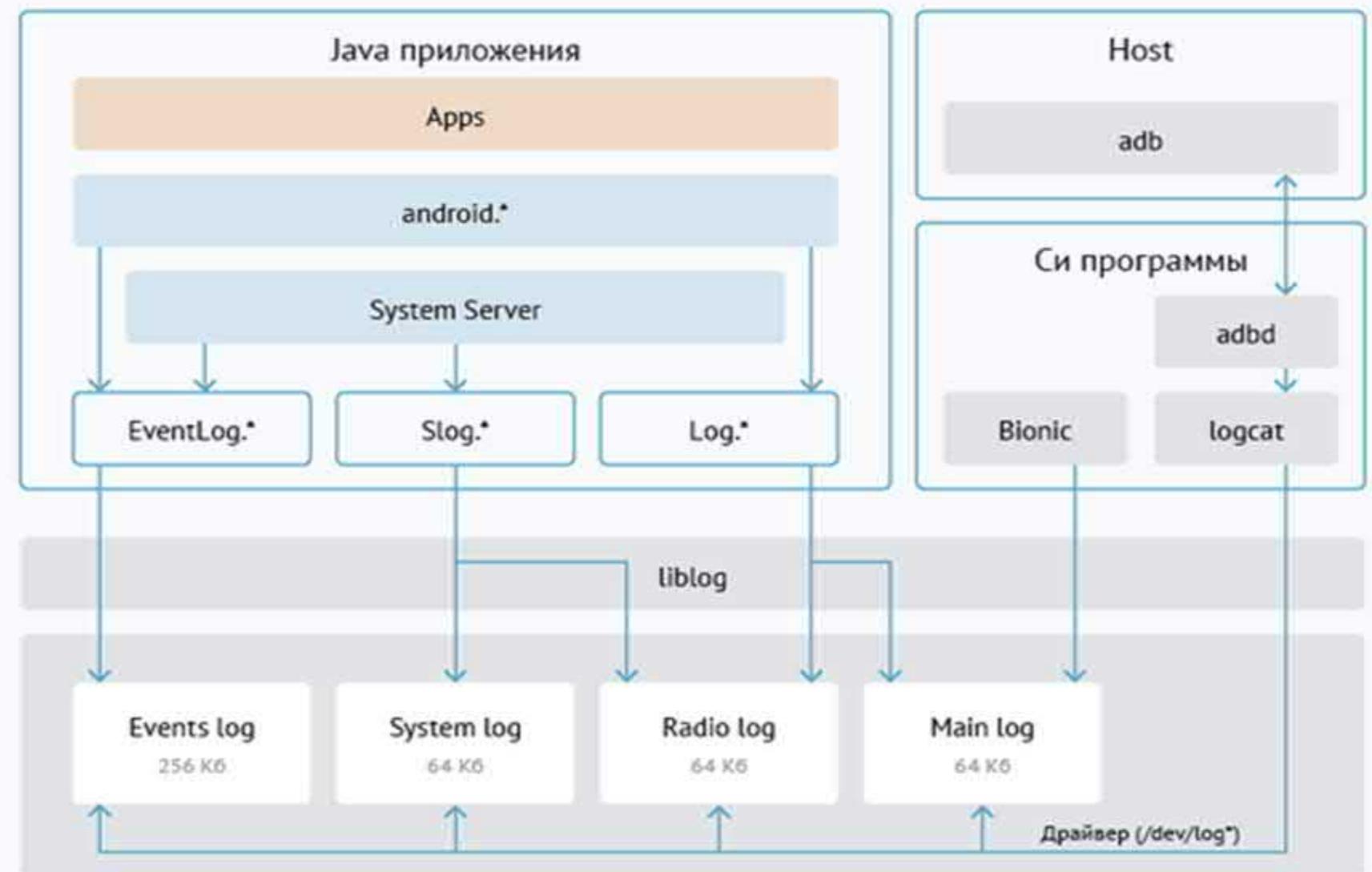
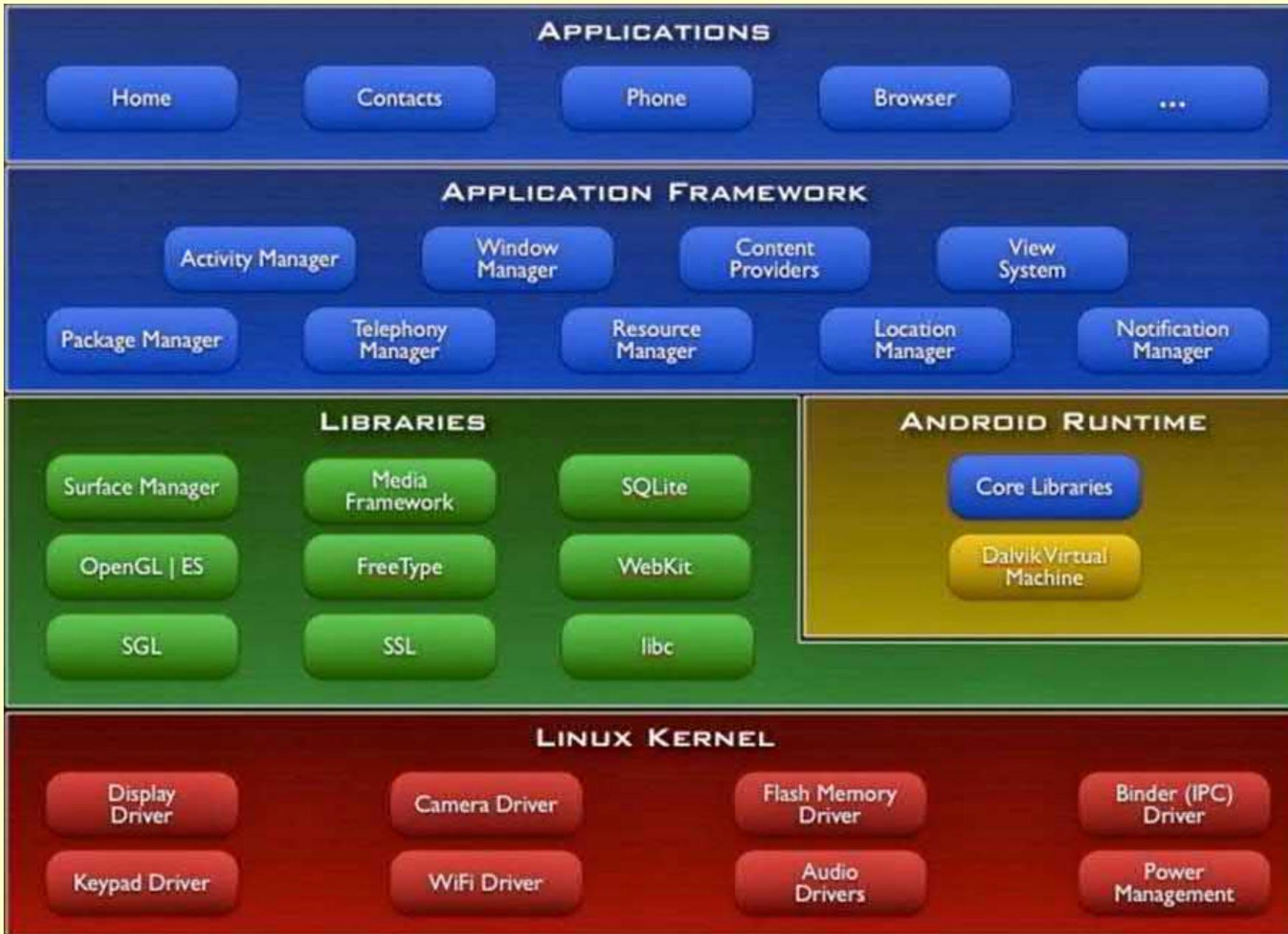


Диаграмма - общая система логирования Андроида

Классический рисунок представляющий архитектуру ОС Android:



Если кому-то сложно с английским, то на всякий случай то же самое по на русском:

Уровень приложений

Встроенные
приложения
(контакты, карты,
браузер и т. п.)

Сторонние
приложения

Приложения
разработчика

Фреймворк приложений

Навигационные
службы

Источники
данных

Оконный
менеджер

Менеджер
деятельностей

Менеджер
пакетов

Телефония

P2P/XMPP

Уведомления

Представления

Менеджер
ресурсов

Библиотеки

Графика
(OpenGL, SGL,
Free Type)

Мультимедиа

SSL & Webkit

libc

SQLite

Менеджер
поверхностей

Библиотеки
Android

Виртуальная
машина Dalvik

Рабочая среда Android

Linux-ядро

Драйверы
оборудования
(USB, экран,
Bluetooth и т. п.)

Управление
питанием

Управление
процессами

Управление
памятью

Если представить компонентную модель Android в виде некоторой иерархии, то в самом низу, как самая фундаментальная и базовая составляющая, будет располагаться ядро операционной системы (Linux Kernel).

Часто компонентную модель ещё называют программным стеком.

Действительно, это определение тут уместно, потому что речь идет о наборе программных продуктов, которые работают вместе для получения итогового результата.

Действия в этой модели выполняются последовательно, и уровни иерархии также последовательно взаимодействуют между собой.

LINUX KERNEL (ЯДРО ЛИНУКС)

Как известно, Андроид основан на несколько урезанном ядре ОС Linux и поэтому на этом уровне мы можем видеть именно его (версии x.x.x).

Оно обеспечивает функционирование системы и отвечает за безопасность, управление памятью, энергосистемой и процессами, а также предоставляет сетевой стек и модель драйверов.

Ядро также действует как уровень абстракции между аппаратным обеспечением и программным стеком.

На предыдущих лекциях мы тщательно изучили ядро Linux и других Unix систем и они в основе своей такие же, как ядро Android.

LIBRARIES (БИБЛИОТЕКИ)

«Выше» ядра, как программное обеспечение промежуточного слоя, лежит набор библиотек (Libraries), предназначенный для обеспечения важнейшего базового функционала для приложений.

То есть именно этот уровень отвечает за предоставление реализованных алгоритмов для вышележащих уровней, поддержку файловых форматов, осуществление кодирования и декодирования информации (в пример можно привести мультимедийные кодеки), отрисовку графики и многое другое.

Библиотеки реализованы на С/C++ и скомпилированы под конкретное аппаратное обеспечение устройства, вместе с которым они и поставляются производителем в предустановленном виде.

Краткое описание некоторых из них:

- Surface Manager – в ОС Android используется композитный менеджер окон, наподобие Compiz (Linux), но более упрощенный.

Вместо того чтобы производить отрисовку графики напрямую в буфер дисплея, система посыпает поступающие команды отрисовки в закадровый буфер, где они накапливаются вместе с другими, составляя некую композицию, а потом выводятся пользователю на экран.

Это позволяет системе создавать интересные бесшовные эффекты, прозрачность окон и плавные переходы.

- Media Framework – библиотеки, реализованные на базе PacketVideo OpenCORE. С их помощью система может осуществлять запись и воспроизведение аудио и видео контента, а также вывод статических изображений. Поддерживаются многие популярные форматы, включая MPEG4, H.264, MP3, AAC, AMR, JPG и PNG.
- SQLite – легковесная и производительная реляционная СУБД, используемая в Android в качестве основного движка для работы с базами данных, используемыми приложениями для хранения информации

- OpenGL | ES –

3D библиотеки — используются для высокооптимизированной отрисовки 3D-графики, при возможности используют аппаратное ускорение.

Их реализации строятся на основе API OpenGL ES 1.0.

3D библиотеки которые используются для высоко оптимизированной отрисовки 3D-графики, при возможности используют аппаратное ускорение.

Их реализации строятся на основе API OpenGL ES 1.0.

OpenGL ES (OpenGL for Embedded Systems) – подмножество графического программного интерфейса OpenGL, адаптированное для работы на встраиваемых системах.

- FreeType – библиотека для работы с битовыми картами, а также для растеризации шрифтов и осуществления операций над ними. Это высококачественный движок для шрифтов и отображения текста.
- LibWebCore – библиотеки известного шустрого браузерного движка WebKit, используемого также в десктопных браузерах Google Chrome и Apple Safari.
- SGL (Skia Graphics Engine) – открытый движок для работы с 2D-графикой. Графическая библиотека является продуктом Google и часто используется в других их программах.
- SSL - библиотеки для поддержки одноименного криптографического протокола.
- Libc – стандартная библиотека языка C, а именно её BSD реализация, настроенная для работы на устройствах на базе Linux. Носит название Bionic.

На этом же уровне располагается Android Runtime – среда выполнения. Ключевыми её составляющими являются набор библиотек ядра и виртуальная машина Dalvik. В 5 версии Android заменена на платформу ART. Библиотеки обеспечивают большую часть низкоуровневой функциональности, доступной библиотекам ядра языка Java.

ANDROID RUNTIME (СРЕДА ВЫПОЛНЕНИЯ АНДРОИД)

Каждое приложение в ОС Android запускается в собственном экземпляре виртуальной машины Dalvik. В 5 версии Android заменена на платформу ART.

Таким образом, все работающие процессы изолированы от операционной системы и друг от друга.

И вообще, архитектура Android Runtime такова, что работа программ осуществляется строго в рамках окружения виртуальной машины.

Благодаря этому осуществляется защита ядра операционной системы от возможного вреда со стороны других её составляющих.

Поэтому код с ошибками или вредоносное ПО не смогут испортить Android и устройство на его базе, когда сработают.

Такая защитная функция, наряду с выполнением программного кода, является одной из ключевых для надстройки Android Runtime.

Dalvik полагается на ядро Linux для выполнения основных системных низкоуровневых функций, таких как, безопасность, потоки, управление процессами и памятью.

Вы можете также писать приложения на C/C++, которые будут работать непосредственно на базовом уровне ОС Linux. Хотя такая возможность и существует, необходимости в этом нет никакой.

Если для приложения важны присущие C/C++ скорость и эффективность работы, Android предоставляет доступ к нативной среде разработки (NDK – Native Development Kit).

Она позволяет разрабатывать приложения на C/C++ с использованием библиотек `libc` и `libm`, а также обеспечивает нативный доступ к OpenGL.

Доступ к устройствам и системным службам Android осуществляется через виртуальную машину Dalvik или платформу ART, которая считается промежуточным слоем.

Благодаря использованию Dalvik для выполнения кода программы разработчики получают в свое распоряжение уровень абстракции, который позволяет им не беспокоиться об особенностях конструкции того или иного устройства.

Виртуальная машина Dalvik может выполнять программы в исполняемом формате DEX (Dalvik Executable).

Данный формат оптимизирован для использования минимального объема памяти.

Исполняемый файл с расширением .dex создается путем компиляции классов Java с помощью инструмента dx, входящего в состав Android SDK.

При использовании IDE Eclipse и плагина ADT (Android Development Tools) компиляция классов Java в формат .dex происходит автоматически

Как было сказано выше, инструмент dx из Android SDK компилирует приложения, написанные на Java, в исполняемый формат (dex) виртуальной машины Dalvik.

Помимо непосредственно исполняемых файлов, в состав приложения Android входят прочие вспомогательные компоненты (такие, например, как файлы с данными и файлы ресурсов).

SDK упаковывает все необходимое для установки приложения в файл с расширением .apk (Android package).

Весь код в одном файле .apk считается одним приложением и этот файл используется для установки данного приложения на устройствах с ОС Android.

APPLICATION FRAMEWORK (КАРКАС ПРИЛОЖЕНИЙ)

Уровнем выше располагается Application Framework, иногда называемый уровнем каркаса приложений.

Именно через каркасы приложений разработчики получают доступ к API, предоставляемым компонентами системы, лежащими ниже уровнем.

Кроме того, благодаря архитектуре фреймворка, любому приложению предоставляются уже реализованные возможности других приложений, к которым разрешено получать доступ.

В базовый набор сервисов и систем, лежащих в основе каждого приложения и являющихся частями фреймворка, входят:

- Activity Manager – менеджер Активностей, который управляет жизненными циклами приложений, сохраняет данные об истории работы с Активностями, а также предоставляет систему навигации по ним.
- Package Manager – менеджер пакетов, управляет установленными пакетами на вашем устройстве, отвечает за установку новых и удаление существующих.
- Window Manager – менеджер окон, управляет окнами, и предоставляет для приложений более высокий уровень абстракции библиотеки Surface Manager.
- Telephony Manager – менеджер телефонии, содержит API для взаимодействия с возможностями телефонии (звонки, смс и т.п.)

- Content Providers – контент-провайдеры, управляют данными, которые одни приложения открывают для других, чтобы те могли их использовать для своей работы.
- Resource Manager – менеджер ресурсов, обеспечивает доступ к ресурсам без функциональности (не несущими кода), например, к строковым данным, графике, файлам и другим.
- View System – богатый и расширяемый набор представлений (Views), который может быть использован для создания визуальных компонентов приложений, например, списков, текстовых полей, таблиц, кнопок или даже встроенного web-браузера.
- Location Manager – менеджер местоположения, позволяет приложениям периодически получать обновленные данные о текущем географическом положении устройства.
- Notification Manager – менеджер оповещений, благодаря которому все приложения могут отображать собственные уведомления для пользователя в строке состояния.

Таким образом, благодаря Application Framework, приложения в ОС Android могут получать в своё распоряжение вспомогательный функционал, благодаря чему реализуется принцип многократного использования компонентов приложений и операционной системы.

Естественно, в рамках политики безопасности.

Стоит отметить, просто на понятийном уровне, что фреймворк лишь выполняет код, написанный для него, в отличие от библиотек, которые исполняются сами.

Ещё одно отличие заключается в том, что фреймворк содержит в себе большое количество библиотек с разной функциональностью и назначением, в то время как библиотеки объединяют в себе наборы функций, близких по логике.

APPLICATIONS (ПРИЛОЖЕНИЯ)

На вершине программного стека Android лежит уровень приложений (Applications). Сюда относится набор базовых приложений, который предустановлен на ОС Android.

Например, в него входят браузер, почтовый клиент, программа для отправки SMS, карты, календарь, менеджер контактов и многие другие.

Список интегрированных приложений может меняться в зависимости от модели устройства и версии Android.

И помимо этого базового набора к уровню приложений относятся в принципе все приложения под платформу Android, в том числе и установленные пользователем.

Считается, что приложения под Android пишутся на языке Java, но нужно отметить, что существует возможность разрабатывать программы и на C/C++ (с помощью Native Development Kit), и на Basic (с помощью Simple) и с использованием других языков.

Также можно создавать собственные программы с помощью конструкторов приложений, таких как App Inventor.

Словом, возможностей тут много.

SoC, драйвера и фрагментация.

Project Treble.

Многоуровневая организация Android.

Механизм(компонента) разделяемой
памяти(Ashmem).

Logger.

Архитектура приложений в Android.

Лекция 9 +

SoC, драйвера и фрагментация

Системы на кристалле

Аппаратное обеспечение традиционных настольных и серверных компьютеров построено вокруг материнской платы, на которую устанавливаются такие важнейшие компоненты «харда», как центральный процессор и оперативная память, и к которой затем подключаются дополнительные платы — например, графическая и сетевая — содержащие остальные компоненты (соответственно, графический процессор и Wi-Fi адаптер).

В отличие от такой схемы, в большинстве Android-устройств используются так называемые системы на кристалле (*system on a chip*, SoC).

Система на кристалле представляет собой набор компонентов компьютера — центральный процессор, блок оперативной памяти, порты ввода-вывода, графический процессор, LTE-, Bluetooth- и Wi-Fi-модемы и т.п. — полностью реализованных и интегрированных в рамках одного микрочипа.

Такой подход позволяет не только уменьшить физический размер устройства, чтобы оно поместилось в кармане, и повысить его производительность за счёт большей локальности и лучшей интеграции между компонентами, но и значительно снизить его энергопотребление и тепловыделение, что особенно актуально для мобильных устройств, питающихся от встроенной батарейки и не имеющих систем активного охлаждения.

Но, конечно, системы на кристалле имеют и свои недостатки.

Наиболее очевидный недостаток состоит в том, что такую систему нельзя «проапгрейдить», докупив, например, дополнительной оперативной памяти, как нельзя и заменить плохо работающий компонент.

Это, в принципе, выгодно компаниям-производителям, поскольку побуждает людей покупать новые устройства, когда существующие морально устаревают или выходят из строя, вместо того, чтобы их точечно обновлять или ремонтировать.

Драйвера

Другой — гораздо более важный именно в контексте Android — недостаток SoC заключается в связанных с ними сложностях с драйверами.

Как и в традиционных системах, основанных на отдельных платах, каждый компонент системы на кристалле — от камеры до LTE-модема — требует для работы использования специальных драйверов.

Но в отличие от традиционных систем, эти драйвера обычно разрабатываются производителями систем на кристалле и специфичны для их конкретных моделей; кроме того, исходные коды таких драйверов обычно не раскрываются, да и бинарные сборки в свободный доступ производителями выкладываться совсем не всегда.

Вместо этого производитель SoC (например, Qualcomm) отдаёт готовую сборку драйверов производителям Android-устройств (например, Sony или LG), которые включают её в свою сборку Android, основанную на коде из Android Open Source Project.

Так и получается, что сборка Android, предустановленная на устройстве производителем, содержит все нужные для этого устройства драйвера, а напрямую использовать сборку для одного устройства на другом невозможно.

В результате авторам сборок Android приходится прилагать отдельные усилия для поддержки каждого семейства моделей Android-устройств.

Сами производители устройств совсем не всегда заинтересованы в том, чтобы выделять ресурсы на портирование своих сборок на новые версии Android.

В то же время разработчикам сторонних дистрибутивов Android приходится извлекать драйвера из сборок, предустанавливаемых производителями, что связано с дополнительными сложностями и не всегда приводит к хорошему результату.

У многих сторонних дистрибутивов хватает ресурсов только для поддержки наиболее популярных моделей устройств

Фрагментация

Это приводит к проблеме, известной как фрагментация: экосистема Android состоит из большого количества различных сборок — как официальных сборок от производителей устройств, так и версий сторонних дистрибутивов.

Многие из них к тому же основаны на старых версиях Android, поскольку для многих устройств обновления сборок от производителя выходят медленно или не выходят совсем.

Конечно, медленное обновление экосистемы означает, что не все пользователи получают небольшие обновления интерфейса и другие видимые пользователю улучшения, которые приносят новые версии Android.

Но оно приносит и две гораздо более серьёзные проблемы.

Во-первых, страдает безопасность.

Хотя современные системы используют продвинутые механизмы защиты, в них постоянно обнаруживаются новые уязвимости.

Обычно эти уязвимости оперативно исправляются разработчиками, но это не помогает пользователям, если обновления системы, содержащие исправления, до них никогда не доходят.

Во-вторых, фрагментация отрицательно сказывается на разработчиках приложений под Android — страдает так называемый developer experience (DX, по аналогии с user experience/UX).

В теории, несмотря на внутренние различия в драйверах и пользовательском интерфейсе разных версий и сборок Android, используемые разработчиками приложений API — Android Framework, OpenGL/Vulkan и другие — должны быть переносимы и работать одинаково.

На практике это, конечно, не всегда так, и разработчикам приходится тестировать и обеспечивать работу своих приложений на множестве версий и сборок Android — на разных устройствах, разных версиях системы, сторонних дистрибутивах и так далее.

Далеко не все производители Android-устройств не считают важным своевременно выпускать обновления для своих устройств.

Например, Google выпускают обновления для своих линеек Nexus и Pixel одновременно с тем, как выходит новая версия Android.

У многих других производителей портирование сборок Android на его новую версию занимает месяцы, но они, тем не менее, стараются выпускать ежемесячные обновления безопасности.

Кроме того, совсем не все устройства, где используется Android, построены на основе SoC.

Android вполне можно установить и на обычный «десктопный» компьютер (подойдут, например, сборки от проектов Android-x86 и RemixOS).

Специальная сборка Android встроена в ChromeOS, что позволяет Chromebook'ам запускать Android-приложения наряду с Linux-приложениями и веб-приложениями.

Аналогичного подхода — запуска специальной сборки Android в контейнере — придерживается проект Anbox, позволяющий использовать Android-приложения на «обыкновенных» Linux-системах. (Напомню, что Android-приложения так легко переносятся на x86-архитектуры, не требуя перекомпиляции, благодаря использованию виртуальной машины Java,

Project Treble

Наиболее прямой способ бороться с фрагментацией — это всячески убеждать производителей устройств не забрасывать поддержку своих сборок Android.

Как показывает практика, это работает, но работает недостаточно хорошо. Было бы гораздо лучше, если можно было бы разработать техническое решение, повышающее переносимость сборок Android и тем самым серьёзно облегчающее задачу авторов сборок и сторонних дистрибутивов.

И такое решение уже существует. В 2017 году Google анонсировали Project Treble — новую, ещё более модульную (по сравнению с уже существующим HAL, hardware abstraction layer) архитектуру взаимодействия драйверов (и остального софта, специфичного для конкретного устройства) с остальными частями системы.

Treble позволяет устанавливать основную систему и драйвера, специфичные для устройства, на разные разделы файловой системы, и обновлять — или как угодно изменять — систему отдельно от установленных драйверов.

Treble в корне меняет ситуацию с медленным выпуском обновлений и плохой переносимостью сборок.

Благодаря Treble устройства от семи разных компаний (Sony, Nokia, OnePlus, Oppo, Xiaomi, Essential и Vivo — а не только от самих Google) смогли участвовать в бета-программе Android Pie.

Treble позволил Essential выпустить обновление до Android Pie для своего Essential Phone прямо в день выхода Android Pie.

Одну и ту же сборку Android — один и тот же бинарный файл без перекомпиляции или каких-либо изменений — теперь можно запускать на любом устройстве, поддерживающем Treble, несмотря на то, что они могут быть основаны на совершенно разных SoC.

Влияние Treble действительно сложно переоценить.

Java принесла возможность «write once, run everywhere» для высокоуровневого кода — в том числе и возможность запускать Android-приложения на компьютерах с практически любой архитектурой процессора. Treble — аналогичный прорыв, позволяющий использовать однажды написанную и скомпилированную сборку Android на устройствах с совершенно разными SoC.

Теперь дело за производителями, которым нужно конвертировать свои драйвера в формат, совместимый с Treble.

Можно надеяться, что через несколько лет проблемы с обновлениями Android-устройств исчезнут окончательно.

Механизм(компонента) разделяемой памяти(Ashmem)

Anonymous Shared Memory (ashmem) — механизм разделяемой памяти. В Линуксе, как правило, данный механизм реализован через POSIX SHM. Разработчики Андроида сочли его недостаточно защищённым, что могло сыграть на руку вредоносному ПО.

Особенностями ashmem-а являются счётчик ссылок, при обнулении которого разделяемая память может быть освобождена (например, память освобождается при завершении всех процессов, использующих её), и сокращение разделяемого региона при нехватке памяти в системе.

Ярким примером использования ashmem-а является процесс zygote, в котором загружается стартовая версия Dalvik VM (или ART) с загруженными базовыми классами и ресурсами, а остальные приложения просто ссылаются на эту память.

Binder имеет ограничение на размер транзакции в 1МБ (иначе будет выброшено исключение TransactionTooLargeException).

Если нам надо передать из одного процесса в другой большой объём данных, можно как раз воспользоваться Ashmem-ом: создать MemoryFile и передать дескриптор файла в другой процесс.

Logger

Обычные дистрибутивы, как правило, используют две системы логирования: лог ядра, доступный через команду `dmesg`, и системные логи, располагающиеся обычно в директории `/var/log`.

Система Андроида включает несколько циклических буферов для хранения сообщений пользовательских программ (что продлевает время жизни карт памяти, так как циклы чтения-записи не расходуются впустую) и не имеет дополнительных задержек от работы с сокетами, которые применяются в стандартном `syslog`-е.

На диаграмме представлена общая система логирования Андроида. Драйвер логирования предоставляет доступ к каждому буферу через `/dev/log/*`. Приложения имеют доступ к ним не напрямую, а через библиотеку `liblog`. С библиотекой `liblog` общаются классы `Log`, `Slog` и `EventLog`. Команда `adb logcat` показывает содержимое буфера “`main`”.

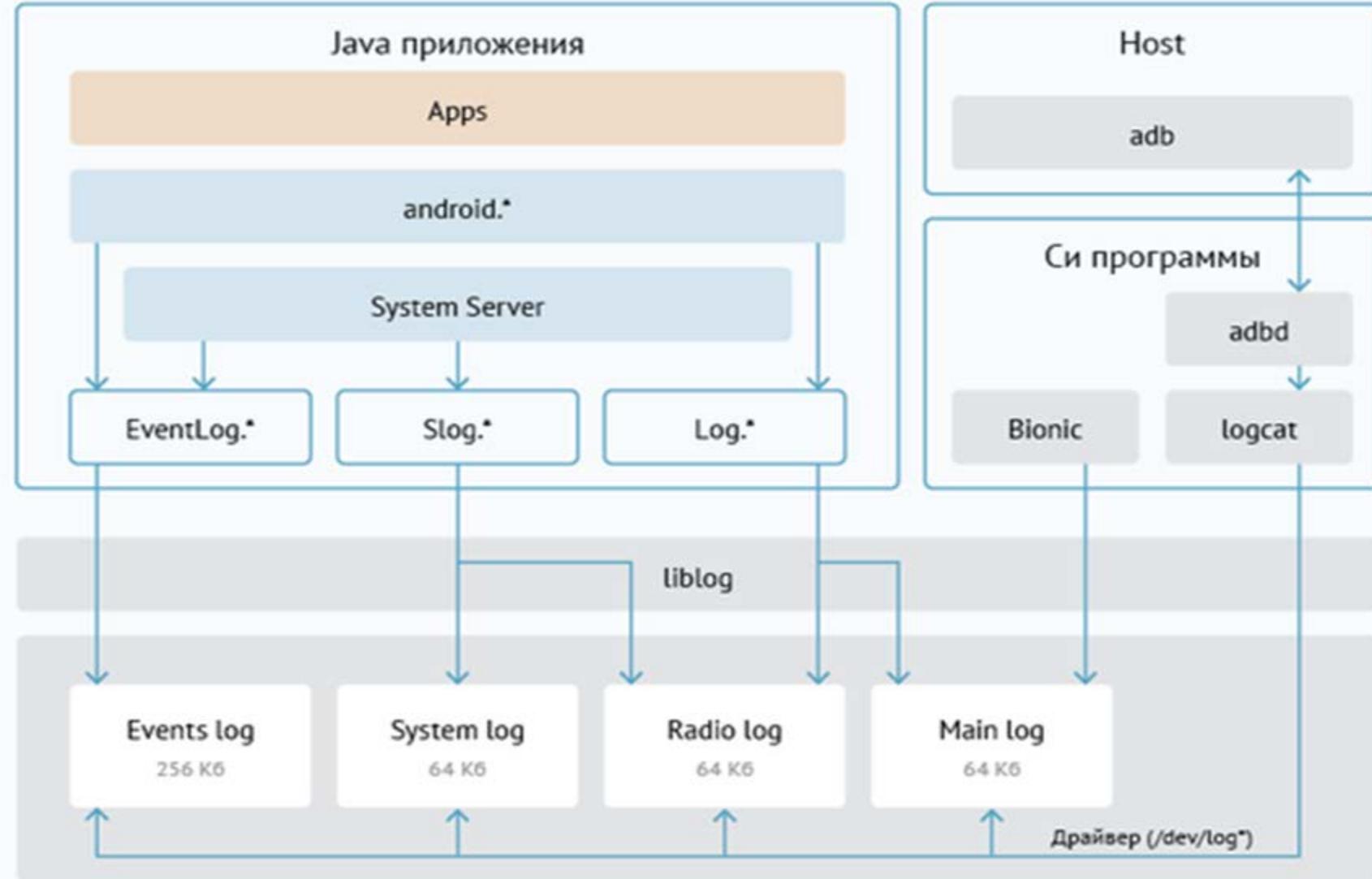
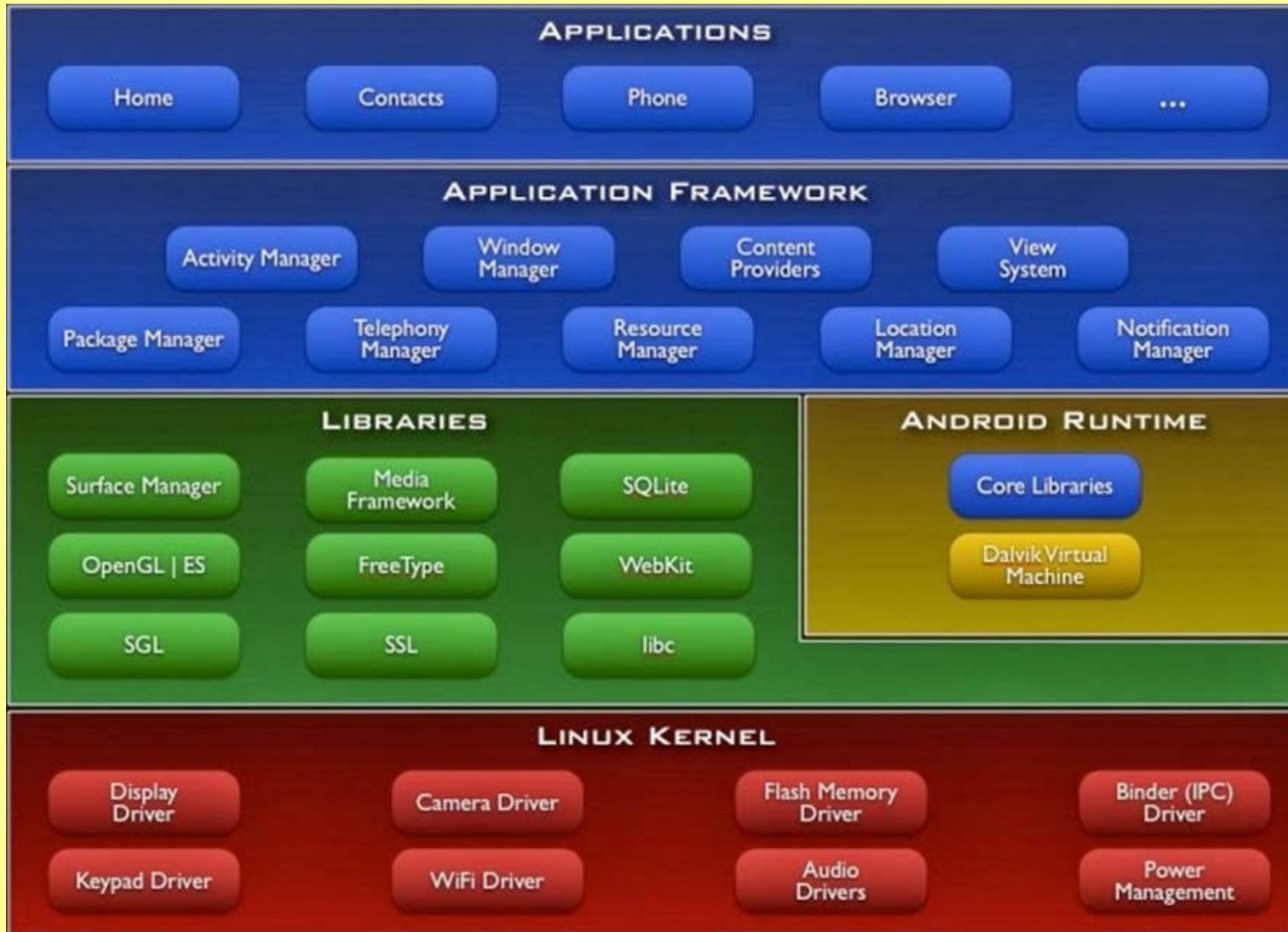
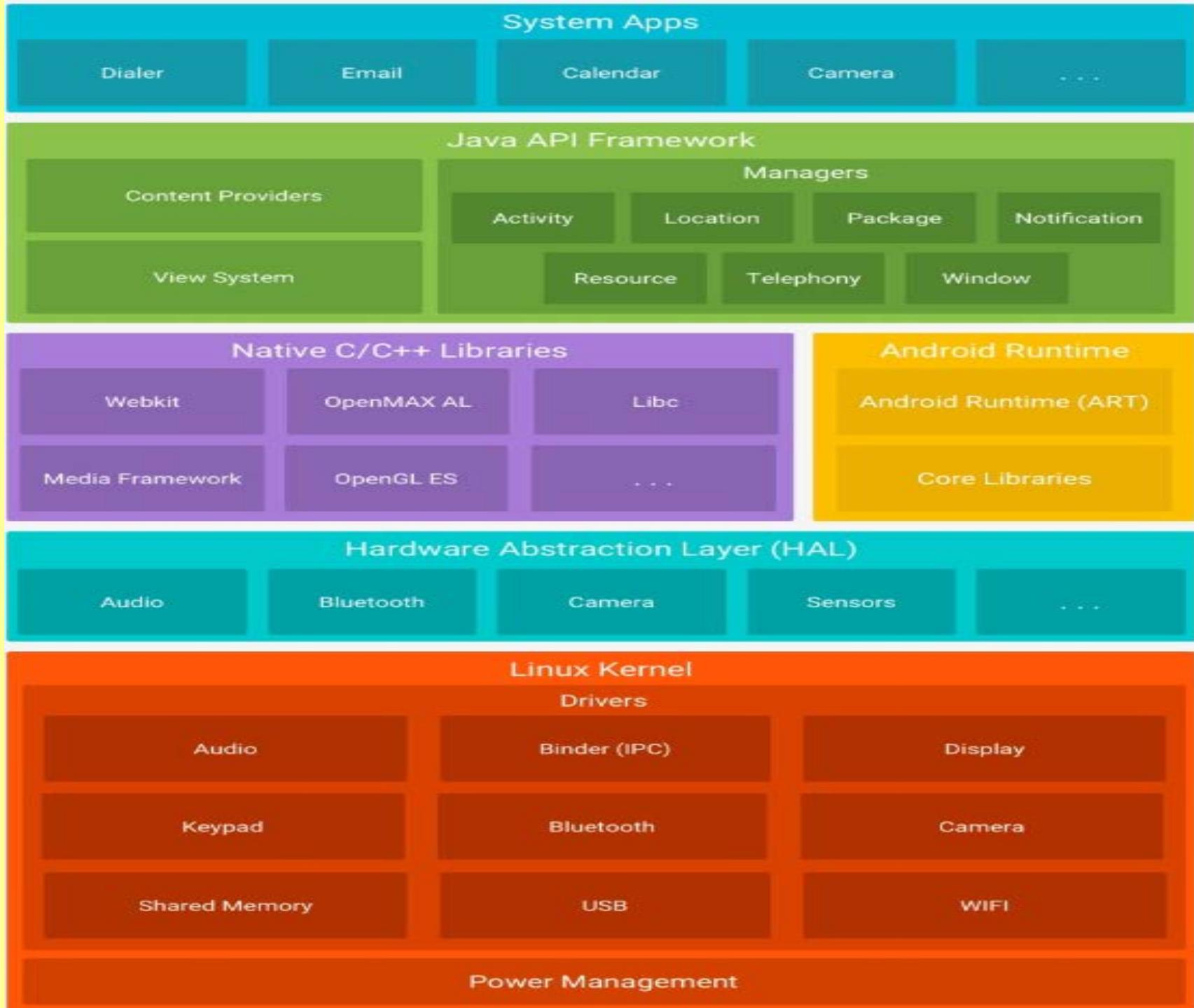


Диаграмма - общая система логирования Андроида

Классический рисунок представляющий организацию ОС Android:



Организация Android версии 5 и выше



Прежде всего, Android HAL создан для "защиты проприетарных" драйверов, предлагаемых некоторыми поставщиками оборудования.

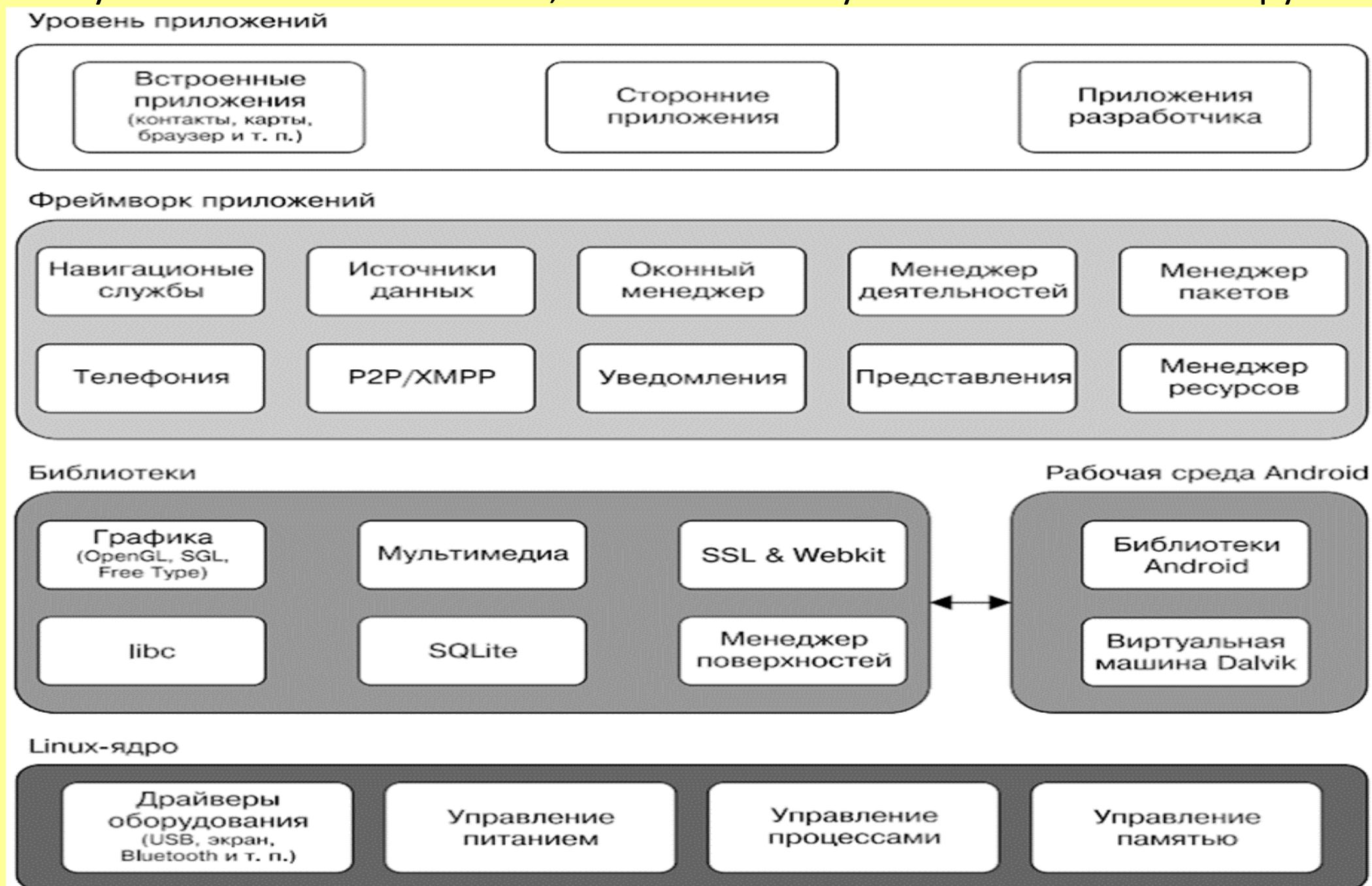
Короче говоря, это делается для того, чтобы избежать лицензии GPL на ядро Linux.

Android помещает действия по управлению оборудованием в пользовательское пространство, а драйвер ядра имеет только простейшую операцию чтения и записи регистров и полностью удаляет различные функциональные операции (такие как логика управления и т. Д.)

Которые могут отражать характеристики оборудования.

Работа Android находится на уровне HAL Android, а Android основан на лицензии Apache, поэтому производители оборудования могут предоставлять только двоичный код, поэтому Android - это только открытая платформа, а не платформа с открытым исходным кодом.

Если кому-то сложно с английским, то на всякий случай то же самое по на русском:



Если представить компонентную модель Android в виде некоторой иерархии, то в самом низу, как самая фундаментальная и базовая составляющая, будет располагаться ядро операционной системы (Linux Kernel).

Часто компонентную модель ещё называют программным стеком.

Действительно, это определение тут уместно, потому что речь идет о наборе программных продуктов, которые работают вместе для получения итогового результата.

Действия в этой модели выполняются последовательно, и уровни иерархии также последовательно взаимодействуют между собой.

LINUX KERNEL (ЯДРО линукс)

Как известно, Андроид основан на несколько урезанном ядре ОС Linux и поэтому на этом уровне мы можем видеть именно его (версии x.x.x).

Оно обеспечивает функционирование системы и отвечает за безопасность, управление памятью, энергосистемой и процессами, а также предоставляет сетевой стек и модель драйверов.

Ядро также действует как уровень абстракции между аппаратным обеспечением и программным стеком.

На предыдущих лекциях мы тщательно изучили ядро Linux и других Unix систем и они в основе своей такие же, как ядро Android.

LIBRARIES (БИБЛИОТЕКИ)

«Выше» ядра, как программное обеспечение промежуточного слоя, лежит набор библиотек (Libraries), предназначенный для обеспечения важнейшего базового функционала для приложений.

То есть именно этот уровень отвечает за предоставление реализованных алгоритмов для вышележащих уровней, поддержку файловых форматов, осуществление кодирования и декодирования информации (в пример можно привести мультимедийные кодеки), отрисовку графики и многое другое.

Библиотеки реализованы на C/C++ и скомпилированы под конкретное аппаратное обеспечение устройства, вместе с которым они и поставляются производителем в предустановленном виде.

Краткое описание некоторых из них:

- Surface Manager – в ОС Android используется композитный менеджер окон, наподобие Compiz (Linux), но более упрощенный.

Вместо того чтобы производить отрисовку графики напрямую в буфер дисплея, система посыпает поступающие команды отрисовки в закадровый буфер, где они накапливаются вместе с другими, составляя некую композицию, а потом выводятся пользователю на экран.

Это позволяет системе создавать интересные бесшовные эффекты, прозрачность окон и плавные переходы.

- Media Framework – библиотеки, реализованные на базе PacketVideo OpenCORE. С их помощью система может осуществлять запись и воспроизведение аудио и видео контента, а также вывод статических изображений. Поддерживаются многие популярные форматы, включая MPEG4, H.264, MP3, AAC, AMR, JPG и PNG.
- SQLite – легковесная и производительная реляционная СУБД, используемая в Android в качестве основного движка для работы с базами данных, используемыми приложениями для хранения информации

- OpenGL | ES –

3D библиотеки — используются для высокооптимизированной отрисовки 3D-графики, при возможности используют аппаратное ускорение.

Их реализации строятся на основе API OpenGL ES 1.0.

3D библиотеки которые используются для высоко оптимизированной отрисовки 3D-графики, при возможности используют аппаратное ускорение.

Их реализации строятся на основе API OpenGL ES 1.0.

OpenGL ES (OpenGL for Embedded Systems) – подмножество графического программного интерфейса OpenGL, адаптированное для работы на встраиваемых системах.

- FreeType – библиотека для работы с битовыми картами, а также для растеризации шрифтов и осуществления операций над ними. Это высококачественный движок для шрифтов и отображения текста.
- LibWebCore – библиотеки известного шустрого браузерного движка WebKit, используемого также в десктопных браузерах Google Chrome и Apple Safari.
- SGL (Skia Graphics Engine) – открытый движок для работы с 2D-графикой. Графическая библиотека является продуктом Google и часто используется в других их программах.
- SSL - библиотеки для поддержки одноименного криптографического протокола.
- Libc – стандартная библиотека языка C, а именно её BSD реализация, настроенная для работы на устройствах на базе Linux. Носит название Bionic.

На этом же уровне располагается Android Runtime – среда выполнения. Ключевыми её составляющими являются набор библиотек ядра и виртуальная машина Dalvik. В 5 версии Android заменена на платформу ART. Библиотеки обеспечивают большую часть низкоуровневой функциональности, доступной библиотекам ядра языка Java.

ANDROID RUNTIME (СРЕДА ВЫПОЛНЕНИЯ АНДРОИД)

Каждое приложение в ОС Android запускается в собственном экземпляре виртуальной машины Dalvik. В 5 версии Android заменена на платформу ART.

Таким образом, все работающие процессы изолированы от операционной системы и друг от друга.

И вообще, архитектура Android Runtime такова, что работа программ осуществляется строго в рамках окружения виртуальной машины.

Благодаря этому осуществляется защита ядра операционной системы от возможного вреда со стороны других её составляющих.

Поэтому код с ошибками или вредоносное ПО не смогут испортить Android и устройство на его базе, когда сработают.

Такая защитная функция, наряду с выполнением программного кода, является одной из ключевых для надстройки Android Runtime.

Dalvik полагается на ядро Linux для выполнения основных системных низкоуровневых функций, таких как, безопасность, потоки, управление процессами и памятью.

Вы можете также писать приложения на C/C++, которые будут работать непосредственно на базовом уровне ОС Linux. Хотя такая возможность и существует, необходимости в этом нет никакой.

Если для приложения важны присущие C/C++ скорость и эффективность работы, Android предоставляет доступ к нативной среде разработки (NDK – Native Development Kit).

Она позволяет разрабатывать приложения на C/C++ с использованием библиотек `libc` и `libm`, а также обеспечивает нативный доступ к OpenGL.

Доступ к устройствам и системным службам Android осуществляется через виртуальную машину Dalvik или платформу ART, которая считается промежуточным слоем.

Благодаря использованию Dalvik для выполнения кода программы разработчики получают в свое распоряжение уровень абстракции, который позволяет им не беспокоиться об особенностях конструкции того или иного устройства.

Виртуальная машина Dalvik может выполнять программы в исполняемом формате DEX (Dalvik Executable).

Данный формат оптимизирован для использования минимального объема памяти.

Исполняемый файл с расширением .dex создается путем компиляции классов Java с помощью инструмента dx, входящего в состав Android SDK.

При использовании IDE Eclipse и плагина ADT (Android Development Tools) компиляция классов Java в формат .dex происходит автоматически

Как было сказано выше, инструмент dx из Android SDK компилирует приложения, написанные на Java, в исполняемый формат (dex) виртуальной машины Dalvik(после 5 версии - ART)

Помимо непосредственно исполняемых файлов, в состав приложения Android входят прочие вспомогательные компоненты (такие, например, как файлы с данными и файлы ресурсов).

SDK упаковывает все необходимое для установки приложения в файл с расширением .apk (Android package).

Весь код в одном файле .apk считается одним приложением и этот файл используется для установки данного приложения на устройствах с ОС Android.

APPLICATION FRAMEWORK (КАРКАС ПРИЛОЖЕНИЙ)

Уровнем выше располагается Application Framework, иногда называемый уровнем каркаса приложений.

Именно через каркасы приложений разработчики получают доступ к API, предоставляемым компонентами системы, лежащими ниже уровнем.

Кроме того, благодаря архитектуре фреймворка, любому приложению предоставляются уже реализованные возможности других приложений, к которым разрешено получать доступ.

В базовый набор сервисов и систем, лежащих в основе каждого приложения и являющихся частями фреймворка, входят:

- Activity Manager – менеджер Активностей, который управляет жизненными циклами приложений, сохраняет данные об истории работы с Активностями, а также предоставляет систему навигации по ним.
- Package Manager – менеджер пакетов, управляет установленными пакетами на вашем устройстве, отвечает за установку новых и удаление существующих.
- Window Manager – менеджер окон, управляет окнами, и предоставляет для приложений более высокий уровень абстракции библиотеки Surface Manager.
- Telephony Manager – менеджер телефонии, содержит API для взаимодействия с возможностями телефонии (звонки, смс и т.п.)

- Content Providers – контент-провайдеры, управляют данными, которые одни приложения открывают для других, чтобы те могли их использовать для своей работы.
- Resource Manager – менеджер ресурсов, обеспечивает доступ к ресурсам без функциональности (не несущими кода), например, к строковым данным, графике, файлам и другим.
- View System – богатый и расширяемый набор представлений (Views), который может быть использован для создания визуальных компонентов приложений, например, списков, текстовых полей, таблиц, кнопок или даже встроенного web-браузера.
- Location Manager – менеджер местоположения, позволяет приложениям периодически получать обновленные данные о текущем географическом положении устройства.
- Notification Manager – менеджер оповещений, благодаря которому все приложения могут отображать собственные уведомления для пользователя в строке состояния.

Таким образом, благодаря Application Framework, приложения в ОС Android могут получать в своё распоряжение вспомогательный функционал, благодаря чему реализуется принцип многократного использования компонентов приложений и операционной системы.

Естественно, в рамках политики безопасности.

Стоит отметить, просто на понятийном уровне, что фреймворк лишь выполняет код, написанный для него, в отличие от библиотек, которые исполняются сами.

Ещё одно отличие заключается в том, что фреймворк содержит в себе большое количество библиотек с разной функциональностью и назначением, в то время как библиотеки объединяют в себе наборы функций, близких по логике.

APPLICATIONS (ПРИЛОЖЕНИЯ)

На вершине программного стека Android лежит уровень приложений (Applications). Сюда относится набор базовых приложений, который предустановлен на ОС Android.

Например, в него входят браузер, почтовый клиент, программа для отправки SMS, карты, календарь, менеджер контактов и многие другие.

Список интегрированных приложений может меняться в зависимости от модели устройства и версии Android.

И помимо этого базового набора к уровню приложений относятся в принципе все приложения под платформу Android, в том числе и установленные пользователем.

Считается, что приложения под Android пишутся на языке Java, но нужно отметить, что существует возможность разрабатывать программы и на C/C++ (с помощью Native Development Kit), и на Basic (с помощью Simple) и с использованием других языков.

Также можно создавать собственные программы с помощью конструкторов приложений, таких как App Inventor.

Словом, возможностей тут много.

Компоненты архитектуры приложений в Android

За время своего существования Android достиг удивительных успехов, став самой установленной мобильной ОС.

За это время было запущено более 14 различных версий операционной системы, причем Android всегда становится все более зрелым.

Тем не менее, очень важная область платформы по-прежнему игнорировалась: стандартный шаблон архитектуры, способный обрабатывать особенности платформы и достаточно простой для понимания и принятия средним разработчиком.

Ну, лучше поздно, чем никогда. На последнем вводе / выводе Google команда Android наконец решила решить эту проблему и ответить на отзывы разработчиков по всему миру, объявив официальную рекомендацию по архитектуре приложений Android и предоставив строительные блоки для ее реализации: новую архитектуру Компоненты.

И что еще лучше, им удалось сделать это, не ставя под угрозу открытость системы, которую мы все знаем и любим.

Lifecycle

ViewModel

LifeData

Room

Компоненты архитектуры пользовательских приложений в Android

Архитектура приложений

Грубо говоря, архитектура приложения представляет собой согласованный план, который необходимо составить до начала процесса разработки.

Этот план предоставляет карту того, как различные компоненты приложения должны быть организованы и связаны друг с другом.

В нем представлены руководящие принципы, которые должны соблюдаться в процессе разработки, и вынуждает жертвовать собой (обычно связанные с большим количеством классов и шаблонов), которые в итоге помогут вам создать хорошо написанное приложение, которое будет более тестируемым, расширяемым и обслуживаемым.

Архитектура прикладного программного обеспечения — это процесс определения структурированного решения, отвечающего всем техническим и эксплуатационным требованиям, при оптимизации общих атрибутов качества, таких как производительность, безопасность и управляемость.

Он включает ряд решений, основанных на широком спектре факторов, и каждое из этих решений может оказать значительное влияние на качество, производительность, ремонтопригодность и общий успех приложения.

Хорошая архитектура учитывает множество факторов, особенно характеристики и ограничения системы.

Существует множество архитектурных решений, каждый из которых имеет свои плюсы и минусы.

Тем не менее, некоторые ключевые понятия являются общими для всех видений.

До последнего ввода-вывода Google система Android не рекомендовала какой-либо конкретной архитектуры для разработки приложений.

- Это означает, что вы были полностью свободны в принятии любой модели:
MVP(Minimal Viable Product (минимально жизнеспособный продукт)),
- MVC(архитектурный паттерн проектирования (Model, View, Controller)),
- MVPP(минимально жизнеспособный продукт, которым мы гордимся) или вообще без паттерна.

Кроме того, платформа Android даже не предоставила нативных решений для проблем, создаваемых самой системой, в частности, жизненным циклом компонента.

Кроме того, платформа Android даже не предоставила нативных решений для проблем, создаваемых самой системой, в частности, жизненным циклом компонента.

Руководство по архитектуре Android определяет некоторые ключевые принципы, которым должно соответствовать хорошее приложение Android, а также предлагает безопасный путь для разработчика для создания хорошего приложения.

Однако в руководстве прямо указано, что представленный маршрут не является обязательным, и в конечном итоге решение является личным.

Разработчик должен решить, какой тип архитектуры выбрать.

Согласно руководству, хорошее Android-приложение должно обеспечивать четкое разделение проблем и управлять пользовательским интерфейсом от модели.

Любой код, который не обрабатывает пользовательский интерфейс или взаимодействие с операционной системой, не должен быть в Деятельности или Фрагменте, потому что поддержание их как можно более чистыми позволит вам избежать многих проблем, связанных с жизненным циклом.

В конце концов, система может уничтожить Деяния или Фрагменты в любое время.

Кроме того, данные должны обрабатываться моделями, которые изолированы от пользовательского интерфейса и, следовательно, от проблем жизненного цикла.

Новая рекомендуемая архитектура приложений под Android

Архитектура, которую рекомендует Android, не может быть легко помечена среди стандартных шаблонов, которые мы знаем.

Он выглядит как шаблон Model View Controller, но он настолько тесно связан с архитектурой системы, что трудно маркировать каждый элемент с использованием известных соглашений.

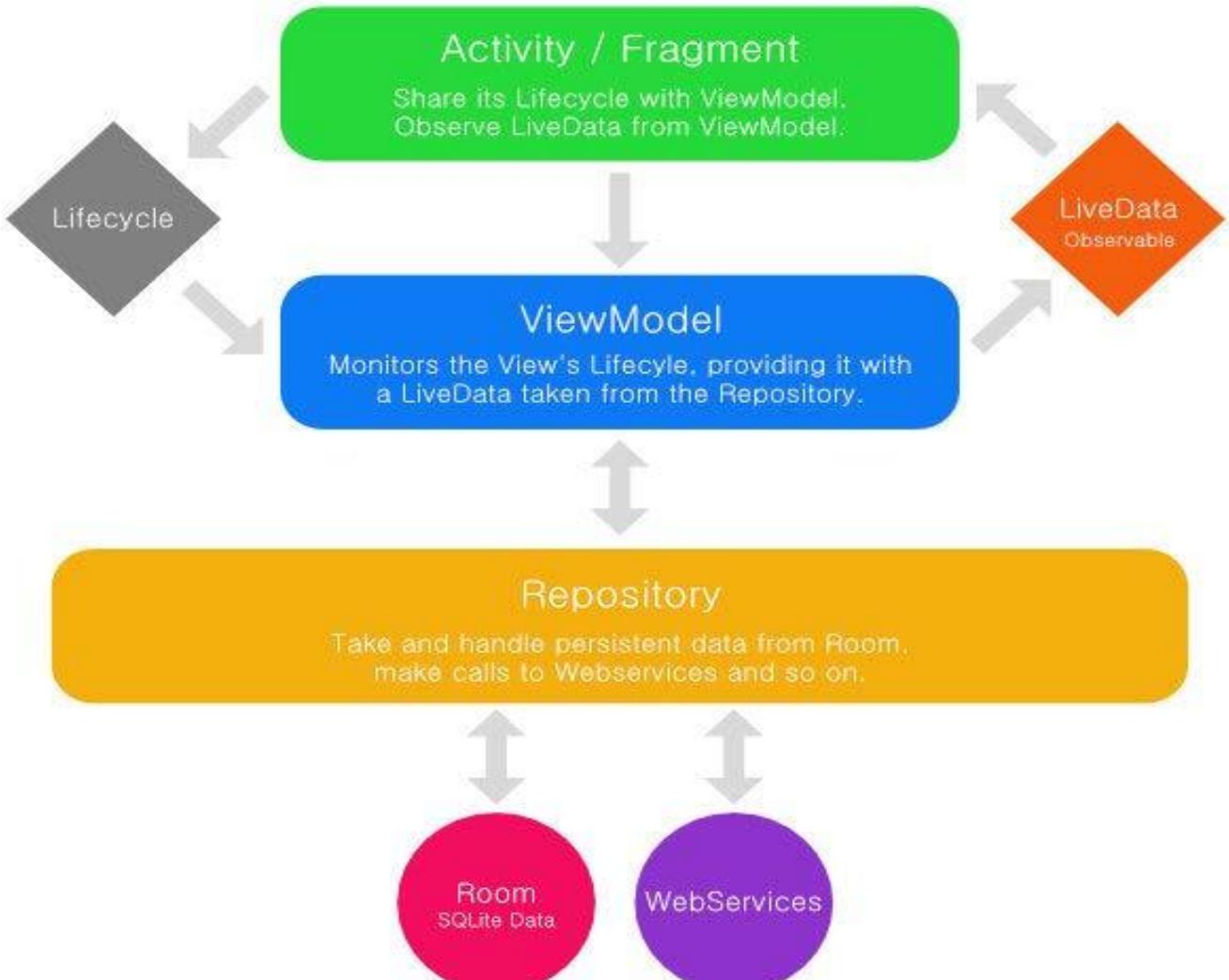
Это, однако, не имеет значения, так как важно то, что он полагается на новые компоненты архитектуры, чтобы создать разделение задач с отличной тестируемостью и ремонтопригодностью.

А еще лучше, это легко реализовать.

Чтобы понять, что предлагает команда Android, мы должны знать все элементы компонентов архитектуры, поскольку именно они сделают за нас тяжелую работу. Существует четыре компонента, каждый с определенной ролью:

- Room ,
- ViewModel ,
- LiveData и
- Lifecycle .

Все эти части имеют свои собственные обязанности, и они работают вместе, чтобы создать надежную архитектуру. Давайте посмотрим на упрощенную схему предложенной архитектуры, чтобы лучше ее понять.



Как видите, у нас есть три основных элемента, каждый из которых несет ответственность.

Activity и Fragment представляют слой View , который не имеет дела с бизнес-логикой и сложными операциями.

Он только настраивает представление, обрабатывает взаимодействие с пользователем и, самое главное, наблюдает за LiveData (элементы LiveData взятые из ViewModel).

ViewModel автоматически наблюдает за состоянием Lifecycle представления, поддерживая согласованность во время изменений конфигурации и других событий жизненного цикла Android.

Это также требуется представлением для извлечения данных из Repository , который предоставляется как наблюдаемые LiveData .

Важно понимать, что ViewModel никогда не ссылается на View напрямую и что обновления данных всегда выполняются сущностью LiveData .

Repository не является специальным компонентом Android.

Это простой класс, без какой-либо конкретной реализации, который отвечает за выборку данных из всех доступных источников, из базы данных в веб-службе.

Он обрабатывает все эти данные, обычно преобразовывая их в наблюдаемые LiveData и делая их доступными для ViewModel .

База данных Room — это библиотека отображений SQLite, которая облегчает процесс работы с базой данных.

Она автоматически записывает кучу шаблонов, проверяет ошибки во время компиляции и, самое главное, может напрямую возвращать запросы с наблюдаемыми LiveData .

Я уверен, что вы заметили, что мы много говорили о наблюдаемых.

Шаблон наблюдателя является одной из основ элемента LiveData и компонентов, LiveData Lifecycle .

Этот шаблон позволяет объекту уведомлять список наблюдателей о любых изменениях его состояния или данных.

Поэтому, когда Activity наблюдает за сущностью LiveData , она будет получать обновления, когда эти данные претерпевают любые изменения.

Еще одна рекомендация Android — консолидировать свою архитектуру с помощью системы Dependency Injection , такой как Google Dagger 2, или с помощью шаблона (который намного проще, чем DI, но без многих его преимуществ).

Мы не будем описывать DI или Service Locator, Однако имейте в виду, что есть некоторые особенности работы с компонентами Dagger 2 и Android, которые полезно изучить.

Компоненты архитектуры приложения

Мы должны глубоко погрузиться в аспекты новых компонентов, чтобы действительно понять и принять эту модель архитектуры.

Однако мы не будем вдаваться во все детали этого урока.

Из-за сложности каждого элемента в этом руководстве мы будем говорить только об общей идее каждого из них и рассмотрим некоторые упрощенные фрагменты кода.

Мы постараемся охватить достаточно места, чтобы представить компоненты и начать работу.

Компоненты, поддерживающие жизненный цикл

К большинству компонентов приложения Android прикреплены жизненные циклы, которые управляются непосредственно самой системой.

До недавнего времени разработчик мог отслеживать состояние компонентов и действовать соответствующим образом, инициализируя и заканчивая задачи в соответствующее время.

Тем не менее, было действительно легко запутаться и сделать ошибки, связанные с этим типом операции.

Но пакет android.arch.lifecycle изменил все это.

Теперь к действиям и фрагментам прикреплен объект Lifecycle который можно наблюдать с помощью классов LifecycleObserver , таких как ViewModel или любой объект, реализующий этот интерфейс.

Это означает, что наблюдатель будет получать обновления об изменениях состояния объекта, который он наблюдает, например, когда действие приостановлено или когда он запускается.

Он также может проверить текущее состояние наблюдаемого объекта.

Так что теперь гораздо проще обрабатывать операции, которые должны учитывать жизненные циклы платформы.

На данный момент, чтобы создать действие или Fragment который соответствует этому новому стандарту, вы должны расширить LifecycleActivity или LifecycleFragment .

Тем не менее, возможно, что это не всегда будет необходимо, так как команда Android стремится полностью интегрировать эти новые инструменты в свою среду.

```
class MainActivity : LifecycleActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main) } }
```

LifecycleObserver получает события LifecycleObserver Lifecycle и может реагировать с помощью аннотации.

Переопределение метода не требуется.

```
class MainActivityObserver : LifecycleObserver, AnkoLogger {  
    @OnLifecycleEvent(Lifecycle.Event.ON_RESUME)  
    fun onResume() {  
        info(«onResume»)  
    }  
    @OnLifecycleEvent(Lifecycle.Event.ON_PAUSE)  
    fun onPause() {  
        info(«onPause») } }
```

Компонент LiveData

Компонент LiveData представляет собой держатель данных, который содержит значение, которое можно наблюдать.

Учитывая, что наблюдатель предоставил Lifecycle во время LiveData экземпляра LiveData , LiveData будет вести себя в соответствии с состоянием Lifecycle .

Если состояние Lifecycle наблюдателя RESUMED, или RESUMED , наблюдатель active ; в противном случае он inactive .

LiveData знает, когда данные были изменены, а также active ли наблюдатель и должен получить обновление.

Еще одна интересная характеристика LiveData заключается в том, что он способен удалять наблюдателя, если он находится в состоянии Lifecycle.State.DESTROYED , избегая утечек памяти при наблюдении операций и фрагментов.

LiveData должен реализовывать onActive и onInactive .

```
lass LocationLiveData(context: Context)
    : LiveData<Location>(), AnkoLogger, LocationListener {
    private val locationManager: LocationManager =
        context.getSystemService(Context.LOCATION_SERVICE) as LocationManager

    override fun onActive() {
        info("onActive")
        locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0f, this)
    }

    override fun onInactive() {
        info("onInactive")
        locationManager.removeUpdates(this)
    }
    // ...
}
```



Чтобы наблюдать компонент LiveData , вы должны вызвать
observer(LifecycleOwner, Observer<T>)

```
class MainActivity : LifecycleActivity(), AnkoLogger {  
    fun observeLocation() {  
        val location = LocationLiveData(this)  
        location.observe(this,  
            Observer { location ->  
                info("location: $location")  
            })  
    }  
}
```

Компонент ViewModel

Одним из наиболее важных классов новых компонентов архитектуры является `ViewModel`, который предназначен для хранения данных, связанных с пользовательским интерфейсом, сохраняя их целостность во время изменений конфигурации, таких как поворот экрана.

`ViewModel` может `LiveData` с `Repository`, получая из него `LiveData` и делая его доступным, чтобы его можно было наблюдать в представлении.

`ViewModel` также не нужно будет делать новые вызовы в `Repository` после изменений конфигурации, что значительно оптимизирует код.

Чтобы создать модель представления, расширьте класс ViewModel .

```
class MainActivityViewModel : ViewModel() {  
  
    private var notes: MutableLiveData<List<String>>?  
    fun getNotes(): LiveData<List<String>> {  
        if (notes == null) {  
            notes = MutableLiveData<List<String>>()  
            loadNotes()  
        }  
        return notes!!  
    }  
    private fun loadNotes() {  
        // do async operation to fetch notes  
    }  
}
```

Для доступа из представления вы можете вызвать
ViewProviders.of(Activity|Fragment).get(ViewModel::class) .

Этот фабричный метод возвратит новый экземпляр ViewModel или получит
сохраненный, в зависимости от ситуации.

```
class MainActivity : LifecycleActivity(), AnkoLogger {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val viewModel = ViewModelProviders.of(this)
            .get(MainActivityViewModel::class.java)
        viewModel.getNotes().observe(
            this, Observer {
                notes -> info(«notes: $notes»)
            }
        )
    }
}
```

Компонент Room

Android поддерживал SQLite с самого начала; однако, чтобы это работало, всегда нужно было писать много шаблонов.

Кроме того, SQLite не сохранял POJO (простые старые объекты Java) и не проверял запросы во время компиляции.

А вот и Room для решения этих проблем!

Это библиотека отображения SQLite, способная сохранять Java POJO, напрямую преобразовывать запросы в объекты, проверять ошибки во время компиляции и создавать наблюдаемые LiveData из результатов запроса.

Room – это библиотека объектно-реляционного сопоставления с некоторыми интересными дополнениями для Android.

До сих пор вы могли делать большую часть того, на что способен Room , используя другие библиотеки ORM Android.

Тем не менее, ни один из них официально не поддерживается, и, самое главное, они не могут давать результаты LiveData .

Библиотека Room идеально подходит в качестве постоянного слоя в предложенной архитектуре Android.

Чтобы создать базу данных Room , вам потребуется @Entity для сохранения, которым может быть любой Java POJO, интерфейс @Dao для выполнения запросов и операций ввода / вывода, а также абстрактный класс @Database который должен расширять RoomDatabase .

```
@Entity
class Note {
    @PrimaryKey
    var id: Long?
    var text: String?
    var date: Long?
}

@Dao
interface NoteDAO {
    @Insert( onConflict = OnConflictStrategy.REPLACE )
    fun insertNote(note: Note): Long

    @Update( onConflict = OnConflictStrategy.REPLACE )
    fun updateNote(note: Note): Int

    @Delete
    fun deleteNote(note: Note): Int

    @Query(`SELECT * FROM note`)
    fun findAllNotes(): LiveData<Note>

    // on Kotlin the query arguments are renamed
    // to arg[N], being N the argument number.
    // on Java the arguments assume its original name
    @Query(`SELECT * FROM note WHERE id = :arg0`)
    fun findNoteById(id: Long): LiveData<Note>
}

@Database(entities = arrayOf(Note::class), version = 1)
abstract class Database : RoomDatabase() {
    abstract fun noteDAO(): NoteDAO
}
```

Добавление компонентов архитектуры в ваш проект

На данный момент, чтобы использовать новые архитектурные компоненты, вам необходимо сначала добавить репозиторий Google в файл build.gradle .

Для получения более подробной информации смотрите официальное руководство .

```
allprojects {  
    repositories {  
        jcenter()  
        // Add Google repository  
        maven { url 'https://maven.google.com' }  
    }  
}
```

Прошивка(firmware) для Android

Первоначально прошивка была термином, используемым для обозначения крошечных критически важных программ, установленных в памяти, доступной только для чтения, или ПЗУ, на электронном устройстве.

Изменение прошивки было либо невозможным, либо требовало специального оборудования, которое обычно было недоступно обычным конечным пользователям.

Однако прошивка для Android отличается от других.

Она включает в себя всю операционную систему Android и хранится в записываемой памяти под флэш-памятью NAND, том же типе памяти, который используется на устройствах хранения, таких как USB-накопители и SD-карты. Слово прошивка используется только потому, что производители устройств не удосужились придумать новое слово для этого.

Прошивку Android также часто называют Android ROM, потому что по умолчанию пользователи не могут напрямую писать в нее.

Иногда прошивку в мобильных устройствах называют Firmware.

Структура прошивки Android

Прошивка, установленная на устройстве Android производителем, содержит сборку операционной системы Android и две дополнительные программы с закрытым исходным кодом, которые обычно незаменимы, загрузчик и радио прошивка.

Загрузчик Android представляет собой небольшой фрагмент проприетарного кода, который отвечает за запуск операционной системы Android, когда устройство включено.

Однако загрузчик почти всегда выполняет еще одну задачу. Он проверяет подлинность операционной системы.

Как он решает, что является подлинным? Он проверяет, был ли загрузочный раздел подписан с использованием уникального ключа OEM, что является сокращением от ключа Original Equipment Manufacturer.

Ключ OEM, конечно же, принадлежит производителю устройства, является закрытым, и вы не можете понять, что это такое.

Из-за проверки подлинности вы не можете напрямую установить пользовательский ПЗУ на устройстве Android.

К счастью, в наши дни большинство производителей устройств позволяют пользователям отключать проверку.

На Android-жаргоне они позволяют пользователям разблокировать загрузчик.

Конкретная процедура, которую вам нужно выполнить, чтобы разблокировать загрузчик, зависит от вашего устройства.

Некоторые производители, такие как Sony и HTC, ожидают, что вы предоставите секретный токен разблокировки.

Другие просто ожидают, что вы запустите фиксированный набор команд с помощью терминала.

Обычно для запуска команд разблокировки используется инструмент fastboot, который является частью Android SDK.

Например, если у вас есть устройство Nexus, вы можете разблокировать его загрузчик, выполнив следующую команду:

```
fastboot flashing unlock
```

Радио прошивка

Это может показаться вам неожиданностью, но ваш смартфон Android фактически запускает две операционных системы на основном процессоре и независимом процессоре, называемом процессором основной полосы.

Радио прошивка относится к операционной системе, которая работает на процессоре основной полосы частот.

Как правило, это RTOS - операционная система реального времени отвечает за управление возможностями сотовой радиосвязи устройства.

Другими словами, это то, что позволяет вашему устройству совершать звонки и подключаться к Интернету с использованием беспроводных технологий, таких как 2G, 3G, 4G LTE, 5G.

RTOS - это проприетарный кусок кода и популярные производители базовых процессоров, такие как Qualcomm, MediaTek и Spreadtrum держат его в секрете. Операционная система Android обычно взаимодействует с RTOS с использованием сокетов и обратных вызовов.

Как правило, замена радио прошивки вашего устройства - плохая идея, так как может привести радио часть смарфона в неисправное состояние.

Билды Android

Android-сборка - это единственная часть прошивки, созданная с открытым исходным кодом.

Следовательно, это единственная часть, которую вы можете изменить и расширить. Когда вы слышите, как энтузиасты Android говорят, что «я сделал новый ПЗУ на своем устройстве», вы можете быть уверены, что речь идет о новой сборке Android.

Сборка Android обычно используется в виде ZIP-файла, который может использоваться fastboot. Он имеет следующее содержимое:

update.zip

```
|-- android-info.txt  
|-- boot.img  
|-- recovery.img  
|-- system.img  
`-- userdata.img
```

android-info.txt

android-info.txt - текстовый файл, определяющий предварительные условия сборки.

Например, он может указывать номера версий загрузчика и радио прошивки, необходимые для сборки.

Вот пример файла android-info.txt:

```
require board=herring
```

```
require version-bootloader=l9020XXJK1
```

```
require version-baseband=l9020XXKD1
```

boot.img

boot.img - это двоичный файл, содержащий как ядро Linux, так и ramdisk в виде архива GZIP.

Ядро - это исполняемый файл zImage для загрузки, который может использоваться загрузчиком.

С другой стороны, ramdisk является файловой системой только для чтения, которая монтируется ядром во время процесса загрузки.

Он содержит хорошо известный процесс init, первый процесс, который запускается любой операционной системой на базе Linux.

Он также содержит различные демоны, такие как adbd и healthd, которые запускаются процессом init.

Вот как выглядит дерево каталогов ramdisk:

ramdisk/

- |-- charger -> /sbin/healthd
- |-- data
- |-- default.prop
- |-- dev
- |-- file_contexts
- |-- fstab.grouper
- |-- init
- |-- init.environ.rc
- |-- init.grouper.rc
- |-- init.grouper.usb.rc
- |-- init.rc
- |-- init.recovery.grouper.rc
- |-- init.trace.rc
- |-- init.usb.rc
- |-- init.zygote32.rc
- |-- proc
- |-- property_contexts
- |-- sbin
 - | |-- adbd
 - | |-- healthd
 - | |-- ueventd -> ../init
 - | `-- watchdogd -> ../init
- |-- seapp_contexts
- |-- selinux_version
- |-- sepolicy
- |-- service_contexts
- |-- sys
- |-- system
- |-- ueventd.grouper.rc
- `-- ueventd.rc

system.img

system.img - образ раздела, который будет установлен в пустой каталог *system*, который вы можете увидеть в приведенном выше дереве.

Он содержит исполняемые файлы, необходимые для запуска операционной системы *Android*.

Он включает в себя системные приложения, шрифты, фреймворки JAR, библиотеки, медиакодеки и многое другое.

Очевидно, что этот файл больше всего интересует пользователей *Android*, когда они запускают новый ПЗУ.

Системный образ также является файлом, который заставляет большинство пользователей *Android* проявлять интерес к пользовательской прошивки.

Файлы системных образов, предоставляемые производителями устройств, часто заполняются ненужными приложениями и настройками, неофициально называемыми *bloatware*.

Единственный способ удалить *bloatware* - заменить образ системы производителя на более желательный образ системы.

userdata.img

userdata.img - образ раздела, который будет установлен в пустой каталог data, который вы можете увидеть в дереве каталогов ramdisk.

Когда вы загружаете пользовательский ПЗУ, этот образ обычно пустой, и он используется для сброса содержимого каталога data.

recovery.img

recovery.img очень похож на **boot.img**.

Он имеет загрузочный исполняемый файл ядра, который может использовать загрузчик, и **ramdisk**.

Следовательно, образ для восстановления также можно использовать для запуска устройства **Android**.

Когда он используется, вместо **Android** запускается очень ограниченная операционная система, которая позволяет пользователю выполнять административные операции, такие как сброс пользовательских данных устройства, установка новой прошивки и создание резервных копий.

Процедура, необходимая для загрузки с использованием образа восстановления, зависит от устройства.

Обычно это включает в себя запуск режима загрузчика, также называемый **fastboot mode**, путем нажатия комбинации аппаратных ключей, присутствующих на устройстве, а затем выбора опции **Recovery**.

Например, на устройстве **Nexus** вам нужно нажать и удерживать кнопку питания в сочетании с кнопкой уменьшения громкости.

Кроме того, вы можете использовать **adb**, инструмент, включенный в **Android SDK**, для прямого входа в режим восстановления.

adb reboot recovery

Использование fastboot

Самый простой способ загрузки новой прошивки на вашем устройстве - использовать инструмент fastboot. fastboot следует протоколу fastboot для связи с устройством Android.

Однако это может произойти только при запуске устройства в режиме fastboot.

Самый быстрый способ войти в режим fastboot - использовать adb:

```
adb reboot bootloader
```

Чтобы запустить пользовательский ПЗУ, который доступен в виде ZIP-файла, содержащего все файлы образов, упомянутые в предыдущем разделе, вы можете использовать команду fastboot update.

Например, вот как вы могли бы загрузить ROM, присутствующий в файле с именем update.zip:

```
fastboot update update.zip
```

Если вы хотите загрузить только определенный образ, вы можете сделать это, используя команду quickboot flash.

Например, вот как вы могли бы загрузить только образ системы:

```
fastboot flash system system.img
```

Аналогичным образом, если вы хотите заменить только загрузочный образ, вы должны использовать следующую команду:

```
fastboot flash boot boot.img
```

Всегда полезно проверить, работает ли загрузочный или восстановительный образ, прежде чем он начнет работать на вашем устройстве.

Для этого вы можете использовать команду `fastboot boot`.

Например, вот как вы можете проверить, совместим ли пользовательский образ восстановления с именем `twrp.img` с вашим устройством:

```
fastboot boot twrp.img
```

Обратите внимание, что ни одна из команд `fastboot`, упомянутых в этом разделе, не будет работать, если загрузчик вашего устройства не был разблокирован.

Полная структура прошивки Android

Рассмотрим состав прошивки , на примере состава прошивки от ZTE , в принципе она с небольшими изменениями идентична на всех Android устройствах

preloader (~0.25 Мб.) - предзагрузчик. Обеспечивает связь телефона с FlashTool-ом в "режиме USB" для прошивки, а также обеспечивает запуск устройства. Грузит в оперативную память uboot и передаёт ему управление.

dsp_b1 (~0.75 Мб.) – Дополнительный микрокод процессора. Обеспечивает исправление аппаратных ошибок микропроцессора и его взаимодействие с набором окружающих микросхем. Порча его превращает телефон в худшем случае в нерабочий гаджет, в лучшем - планшет без коммуникаций. Восстановление будет очень проблематичным.

nvram (~3.0 Мб.) - хранит калибровки железок, IMEI, MAC-адреса BT и WIFI и другое. Точка монтирования /data/nvram.

seccnfg (~0,125Мб.) - обычно содержит только "FF FF...".

uboot (~0.375 Мб.) - загрузчик операционной системы + драйверы для инициализации основного оборудования (дисплей, процессор, GPIO), необходимые для обеспечения загрузки ОС.

boot (~6.0 Мб.) - ядро и драйверы операционной системы (камеры, датчики, сенсоры). Точка монтирования /.

recovery (~6.0 Мб.) - минисистема (система в ядре) функцией которой является только резервирование/восстановление приложений системы, сброс до заводских установок. В расширенном recovery функционал конечно же намного богаче.

secstatic (~1.156 Мб.) - sec_ro , этот раздел занимает хороший кусок ROM и в нём находятся службы Google. Точка монтирования /system/secro. Файловая система yaffs2.

misc (~0.375 Мб.) - обычно содержит только "FF FF...".

logo (~3.0 Мб.) - Первая картинка при включении, картинка зарядки. На 95% содержит только "FF FF...".

expdb (~0.65 Мб.) - обычно содержит только "FF FF...".

system (~160-210 Мб.) - системный раздел Android.

Тут всё что относится к функционированию аппарата, от интерфейса до поддерживаемых функций операционной системы.

Всё, что здесь изменяется не подлежит восстановлению заводским сбросом. Точка монтирования /system. Файловая система yaffs2.

cache (~62.0 Мб.) - раздел для расположения временных файлов. Обычно используется приложениями ("Маркет", "ROM Manager" ...). При утрате содержимого раздела функционирование системы не пострадает. Неверное же содержимое может привести к зависанию при загрузке устройства.

Полностью стирается при заводском сбросе.

Точка монтирования /cache. Файловая система yaffs2.

userdata (~220-290 Мб.) - data, это раздел для установки программ календарей, телефонок, профилей, настроек различных программ и системы. При утрате содержимого раздела обычно функционирование системы не страдает.

Неверное же содержимое может привести к зависанию при загрузке устройства.

Полностью стирается при заводском сбросе.

Точка монтирования /data. Файловая система yaffs2.

Пример разметки NAND

```
[PART] blksz: 2048B
[PART] [0x0000000000000000-0x00000000003FFF] "PRELOADER" (128 blocks)
[PART] [0x000000000040000-0x0000000000FFFF] "DSP_BL" (384 blocks)
[PART] [0x0000000000100000-0x00000000003FFF] "NVRAM" (1536 blocks)
[PART] [0x0000000000400000-0x000000000041FFF] "SECCNFG" (64 blocks)
[PART] [0x0000000000420000-0x000000000047FFF] "UBOOT" (192 blocks)
[PART] [0x0000000000480000-0x000000000097FFF] "BOOTIMG" (2560 blocks)
[PART] [0x0000000000980000-0x0000000000E7FFF] "RECOVERY" (2560 blocks)
[PART] [0x0000000000E80000-0x0000000000F9FFF] "SECSTATIC" (576 blocks)
[PART] [0x0000000000FA0000-0x0000000000FFFFF] "MISC" (192 blocks)
[PART] [0x0000000000100000-0x000000000012FFF] "LOGO" (1536 blocks)
[PART] [0x00000000001300000-0x0000000000139FFF] "EXPDB" (320 blocks)
[PART] [0x000000000013A0000-0x0000000000FE9FFF] "ANDSYSIMG" (120320 blocks)
[PART] [0x0000000000FEA0000-0x000000000011C9FFF] "CACHE" (15360 blocks)
[PART] [0x000000000011CA0000-0x000000000011C9FFF] "USER" (0 blocks)
```

Preloader

Процесс загрузки протекает так: preloader -> uboot -> kernel. Соответственно, preloader делает начальную инициализацию, читает разметку NAND и совершает прыжок по адресу uboot. В зависимости от причины включения (обычное, воткнут кабель, ...), uboot продолжает загрузку или нет.

Preloader отвечает за отрисовку индикации заряда и логотипа, пока uboot решает, что ему делать.

Preloader также отвечает за прошивку NAND при помощи утилиты SP Flash Tool, которая используется на производстве.

uboot

Das U-Boot - это главный загрузчик с открытым исходным кодом, используемый для упаковки инструкций во встроенные устройства для загрузки ядра операционной системы устройства.

Он часто используется для загрузки и загрузки ядер и файловых систем Linux на встроенных устройствах с архитектурой ARM.

U-Boot поддерживает запуск интерфейса командной строки через последовательный порт.

Используя командную строку, пользователи могут загружать и загружать ядро, возможно, изменяя параметры по умолчанию.

Также есть команды для чтения информации об устройстве, чтения и записи во флэш-память, загрузки файлов (ядра, загрузочного образа и т. д.)

Из последовательного порта или сети, управления деревом устройств и использования переменных среды (которые могут быть записаны в постоянное хранилище) для Управляйте поведением U-Boot, например командой загрузки по умолчанию и тайм-аутом перед автоматической загрузкой, а также данными оборудования, такими как MAC-адрес Ethernet.

Структура и назначение папок и файлов Android

/dev/ — в данном разделе содержится информация о устройствах системы и файлов.

раздел **/data/** — пользовательский раздел в котором находятся установленные приложения, личные настройки

папка **/data/app** — здесь находятся установленные приложения, игры.

папка **/data/app-lib** — дополнительные библиотеки необходимые для работы определенных приложений (присутствует в новых версиях Android).

папка **/data/dalvik-cache** — кеш-память, для работы Java машины

папка **/data/data** — в данной папке находятся индивидуальные настройки каждого пользовательского приложения, библиотеки и другие файлы необходимые файлы для их работы.

папка **/data/system/** — в данном разделе находятся глобальные настройки пользовательского окружения, синхронизация, аккаунты, блокировка.

файлы **gesture.key**, **locksettings.db**, **locksettings.db-shm**, **locksettings.db-wal** — графический ключ, пин-код.

раздел **/efs/** — находится файлы и папки отвечающие за IMEI (данный раздел имеется не во всех Android).

раздел **/preload/** — в данном разделе находятся дополнительные файлы и папки, которые зеркалируются в раздел **/system/** (данный раздел имеется не во всех Android, преимущественно в Samsung). Там могут находиться дополнительные приложения или фоновые рисунки.

раздел **/system/** — данный раздел содержит системные папки и файлы необходимые для функционирования Android.

папка **/system/app** — здесь находятся системные приложения и сервисы (в новых ОС Android сервисные приложения вынесли в другую папку `priv-app`).

папки **/system/bin** и **/system/xbin** — папка содержит файлы и ссылки на исполняемые бинарные файлы.

файл **/system/xbin/su** — файл отвечающий за Root права.

папка **/system/camerdata** — в данной папке находятся файлы отвечающие за работу камеры.

папка **/system/etc** — в данной папке находятся конфигурационные файлы необходимые при загрузке ОС а также необходимые в процессе работы различных программ.

папка **/system/init.d** — в данной папке находятся скрипты, которые могут влиять на работу системы.

файл **/system/etc/ hosts** — файл отвечающий за блокировку, переадресацию веб адресов.

файл **/system/etc/ apns.conf** — файл с информацией о точках доступах интернет (APN).

файл **/system/etc/gps.conf** — настройки GPS.

папка **/system/fonts** — папка с системными шрифтами.

папка **/system/framework** — папка с «процессами» Android.

папка **/system/lib/** — библиотеки системных приложений и сервисов.

папка **/system/lib/modules** — драйверы системы.

папка **/system/media** — папка с системными звуками и анимацией включения.

файл **/system/media/bootanimation.zip** — исполняемый архив с загрузочной анимацией.

папка **/system/priv-app** — папка с сервисами/приложениями Android.

файл **/system/build.prop** — конфигурационный файл с помощью которого можно изменить системные настройки, например модель устройства.

файл **/vendor/build.prop** — конфигурационный файл, в котором описываются специфические настройки для конкретного устройства.

Здесь описываются основные папки, входящие в состав образа `system.img`. В зависимости от модели устройства, производитель может добавлять свои папки.

Основы Android приложений. Компоненты приложения Манифест приложения.

Лекция 10 +

Рассматриваемые вопросы

Основы Android приложений.

Компоненты приложения.

Явления (Activities).

Сервисы (Services).

Поставщики содержимого.

Широковещательные приемники.

Запуск компонентов.

Намерение (intent). Файл манифеста.

Описание компонентов.

Описание возможностей компонентов.

Описание требований приложения. Ресурсы приложения

Android предоставляет богатый фреймворк, который позволяет разрабатывать современные приложения и игры для мобильных устройств на языке программирования Java.

Рассмотрим детальную информацию о том, как создавать приложения с использованием различных API.

Важно, чтобы вы поняли следующие фундаментальные концепции фреймворка:

- Приложения имеют несколько точек входа
- Приложения адаптированы под различные устройства

Концепция-Приложения имеют несколько точек входа

Приложения Android состоят из набора различных компонентов, который могут работать обособленно.

Например, отдельное явление предоставляет экран с элементами пользовательского интерфейса, а сервисы независимо выполняют фоновую работу.

Из одного компонента можно запустить другой, используя намерения.

Вы даже можете запустить компоненты другого приложения.

Благодаря такой модели, существует множество точек входа в приложение, а также существует возможность использовать приложения по умолчанию для некоторых действий.

Концепция - Приложения адаптированы под различные устройства

Android предоставляет инструменты для разработки индивидуальных ресурсов под каждую конфигурацию устройств.

Например, вы можете создавать различную XML разметку для устройств с различными размерами экрана и система автоматически будет выбирать нужную.

Вы можете запрашивать доступность функций на устройстве во время выполнения приложения, например узнать, есть ли на нем камера.

Если это необходимо, вы также можете описать зависимость приложения от функций и запретить установку на устройства, которые эти функции не поддерживают

Основы Android приложений

Android приложения написаны на языке программирования Java.

Инструменты Android SDK компилируют ваш код — включая все данные и ресурсы — в файл APK (Android Package): пакет Android, который является архивом файлов с расширением .apk.

Один APK файл включает все содержимое отдельного Android приложения и этот файл Android устройства используют для установки приложения.

После установки на устройство, каждое приложение работает в его собственной защищенной песочнице:

- Операционная система Android – это многопользовательская система Linux, в которой каждое приложение запускается под отдельным пользователем.
- По умолчанию, система присваивает каждому приложению уникальный идентификатор пользователя Linux (этот ID используется системой и неизвестен приложению). Система устанавливает разрешения для всех файлов приложения и только пользователь с таким ID может иметь к ним доступ.
- Каждый процесс выполняется в своей собственной виртуальной машине, поэтому приложение выполняется изолировано от остальных приложений.
- По умолчанию, каждое приложение запускается в своем собственном отдельном процессе. Android запускает процесс, когда требуется выполнить компонент приложения и останавливает процесс, когда компонент больше не требуется, или если системе недостаточно памяти для выполнения других приложений.

Таким образом, в системе Android реализован принцип наименьших привилегий.

Так что по умолчанию каждое приложение имеет доступ только к компонентам, которые необходимы ему для работы и не более того.

Благодаря этому создается безопасная среда, в которой приложение не может получить доступ к элементам системы, для которых не получено соответствующее разрешение.

Однако, существуют способы обмена данными между приложениями и доступа к системным сервисам:

- Возможно установить для двух приложений общий идентификатор, в этом случае они смогут получить доступ к файлам друг друга. Для экономии ресурсов системы, приложения с одинаковым идентификатором могут также запускаться в одном и том же процессе и использовать одну и ту же виртуальную машину (приложения при этом должны быть подписаны одним и тем же сертификатом).
- Приложение может запросить разрешение на доступ к данным, например контактам, SMS сообщениям, присоединенным хранилищам (SD карты), камере, Bluetooth и другим. Все разрешения должны быть подтверждены пользователем во время установки приложения

Это основы того, как Android приложение существует в системе. Остальная часть текущего материала познакомит вас:

- С базовыми компонентами фреймворка, которые определяют ваше приложение.
- С файлом манифеста, в котором описаны компоненты и требования к устройству для вашего приложения.
- С ресурсами, которые хранятся отдельно от кода и позволяют изящно оптимизировать ваше приложение для различных конфигураций Android устройств.

Компоненты приложения

Компоненты являются основными строительными блоками Android приложений. Каждый компонент является отдельной точкой входа в ваше приложение.

Не все компоненты являются фактически точками входа для пользователя, некоторые из них зависят друг от друга, но каждый из компонентов представляет из себя уникальный строительный блок, который помогает определить поведение приложения в системе.

Существует четыре разных типа компонентов.

Каждый из них служит для различных целей, имеет свой собственный жизненный цикл, который определяет, как компонент создается и уничтожается.

Вот эти четыре типа:

Явления (Activities)

Явления представляют собой единый экран с пользовательским интерфейсом. Например, почтовый клиент может использовать одно явление для отображения списка входящих писем, другое явление для чтения конкретного письма, а третье явление для написания нового сообщения.

И хотя все явления работают вместе, чтобы сформировать общую функциональность приложения, каждое из них независимо от других. Кроме того, приложения могут запускать явления друг друга (если это разрешено).

Например, приложение фотокамера может запустить явление почтового клиента для создания нового письма с только что отнятой фотографией в качестве вложения.

Явления реализуются как подклассы базового класса `Activity`, о них вы узнаете в последующих материалах.

Сервисы (Services)

Сервис, это компонент, работающий в фоновом режиме и предназначенный для выполнения длительных операций или удаленных процессов.

Сервис не предоставляет пользовательский интерфейс.

Например, сервис может проигрывать музыку на заднем фоне, пока пользователь находится в другом приложении, или может закачивать данные по сети, не мешая пользователю взаимодействовать с устройством.

Другие компоненты, например явления, могут запускать сервисы для отдельной работы или могут взаимодействовать с ними.

Службы реализуются как подклассы базового класса `Service`, о них вы узнаете в последующих материалах.

Поставщики содержимого

Поставщики содержимого управляют общими данными в приложении.

Вы можете хранить данные в файловой системе, в базе данных SQLite, в сети, в любом другом постоянном хранилище, к которому приложение имеет доступ.

Через поставщики содержимого, другие приложения могут получить или модифицировать данные вашего приложения (если это разрешено).

Например, Android предоставляет поставщик содержимого для управления контактными данными.

Таким образом, любое приложение, которое имеет соответствующее разрешение, может запросить поставщики содержимого (например `ContactsContract.Data`) для чтения или изменения информации о каком-либо человеке.

Поставщики содержимого также удобны для чтения и записи не открытых данных приложения.

Например, приложение NotePad использует поставщики содержимого для сохранения записей.

Поставщики содержимого реализуются как подклассы базового класса `ContentProvider` и должны содержать стандартный набор методов, которые позволяют другим приложениям выполнять транзакции.

Подробную информацию узнаете в последующих материалах.

Широковещательные приемники

Широковещательный приемник, это компонент, который реагирует на общесистемное оповещение.

Многие широковещательные рассылки создает система — например, уведомление о выключении экрана, о низком заряде батареи, о снятой фотографии.

Приложения также могут создавать рассылки — например, уведомление об окончании загрузки файла, чтобы другие приложения могли его использовать.

Хотя широковещательные приемники не имеют пользовательского интерфейса, они могут создавать уведомления в строке состояния, чтобы предупредить пользователя о происходящем событии.

Однако чаще всего широковещательный приемник является просто мостом между другими компонентами и делает лишь малую часть работы.

Например, это может быть запуск сервиса для выполнения работы, основанной на событии.

Широковещательные приемники реализуются как подклассы базового класса `BroadcastReceiver` и каждая рассылка передается как объект типа `Intent`.

Подробную информацию узнаете в последующих материалах.

Уникальной особенностью Android является возможность запуска одним приложением компонентов другого.

Например вы хотите сделать фотографию.

Наверняка на устройстве уже есть приложение, которое это умеет делать, и вы можете использовать его, вместо того, чтобы разрабатывать подобный функционал заново.

Вам не нужно добавлять исходный код другого приложения и даже ссылки на него.

Вместо этого вы просто запускаете явление приложения Камера и делаете фотографию.

После спуска затвора, фотография будет передана в ваше приложение и вы сможете ее использовать.

Для пользователя будет казаться, что работа с камерой — это часть вашего приложения.

Когда система запускает компонент, запускается процесс для приложения (если оно не было уже запущено) и создаются экземпляры классов, необходимые для работы компонента.

Например, если ваше приложение запустило явление другого приложения, это явление будет работать в процессе другого приложения, а не вашего.

Поэтому, в отличие от приложений в большинстве других систем, Android приложения не имеют единой точки входа в программу (например, нет основной функции `main()`).

Поскольку система запускает каждое приложение в отдельном процессе с разрешением для файлов, которое ограничивает доступ других приложений, ваше приложение не может напрямую активировать компоненты другого приложения.

Но это может сделать система Android.

Таким образом, чтобы запустить компонент другого приложения, вы должны передать в систему сообщение, которое определяет ваше намерение для запуска конкретного компонента.

И система запустит этот компонент для вас.

Запуск компонентов

Три из четырех типов компонентов — явления, сервисы и широковещательные приемники — запускаются с помощью асинхронного сообщения, которое называется намерение (*intent*).

Намерения связывают отдельные компоненты друг с другом во время выполнения (вы можете думать о них как о посыльных, которые запрашивают действия у других компонентов), независимо от того, принадлежит ли компонент вашему приложению или какому-либо другому.

Намерения создаются с помощью объектов типа Intent, которые определяют сообщение для запуска конкретного компонента или компонента заданного типа. Такие намерения называются соответственно явные и неявные.

Для явлений и сервисов, намерения определяют действие для выполнения (например, «что-то показать» или «что-то передать») и могут содержать URI данных. Например, намерение может передать запрос явлению, чтобы последнее показало изображение или веб-страницу.

В некоторых случаях, вы можете запустить явление, чтобы получить от него результат.

В таких случаях, явление возвращает результат так же в объекте Intent (например, вы можете оформить намерение для выбора контакта, и вам вернется другое намерение, содержащее URI, который указывает на выбранный контакт).

Для широковещательных приемников, намерения просто описывают сообщение, которое будет разослано (например, рассылка о низком заряде батареи может содержать только строку «батарея разряжена»).

Последний тип компонентов, поставщики содержимого, не могут быть запущены с помощью намерения.

Они активируются, когда получают запрос от объекта типа ContentResolver. Этот объект обрабатывает все транзакции напрямую, используя поставщики содержимого и методы класса ContentResolver.

Это оставляет слой абстракции между поставщиками содержимого и компонентом, запросившим информацию (для безопасности).

Вот различные методы для запуска каждого типа компонентов:

- Вы можете запустить явление (или заставить его сделать что-то новое), передав объект намерения Intent в метод `startActivity()` или `startActivityForResult()` (если вы хотите, чтобы явление вернуло результат работы).
- Вы можете запустить сервис (или передать новые инструкции на уже работающий), передав объект намерения Intent в метод `startService()`. Или можете связать сервис передав Intent в метод `bindService()`.
- Вы можете инициировать рассылку передав намерение в методы `sendBroadcast()`, `sendOrderedBroadcast()` или `sendStickyBroadcast()`.
- Вы можете выполнять запросы к поставщикам содержимого с помощью метода `query()` класса `ContentResolver`.

Подробную информацию узнаете в последующих материалах

Описание компонентов

Первичной задачей файла манифеста является информирование системы о компонентах приложения.

Например, манифест может описывать явление следующим образом:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest ... >
3   <application android:icon="@drawable/app_icon.png" ... >
4     <activity android:name="com.example.project.ExampleActivity"
5       android:label="@string/example_label" ... >
6     </activity>
7   ...
8   </application>
9 </manifest>
```

Атрибут `android:icon` элемента `<application>` ссылается на ресурс, в котором хранится иконка приложения.

Атрибут `android:name` элемента `<activity>` содержит полное имя класса явления (он является наследником базового класса `Activity`), а атрибут `android:label` содержит заголовок явления, который будет показан пользователю.

Вам нужно описать все компоненты приложения, используя следующие теги:

- `<activity>` для явлений
- `<service>` для сервисов
- `<receiver>` для широковещательных приемников
- `<provider>` для поставщиков содержимого

Если вы создадите в коде явления, сервисы или поставщики содержимого, но не объявили их в файле манифеста, они будут невидимы для системы и не смогут быть запущены.

Однако, широковещательные приемники могут быть либо объявлены в манифесте, либо созданы в коде динамически (объекты типа `BroadcastReceiver`) и зарегистрированы в системе с помощью метода `registerReceiver()`.

Подробную информацию о структуре файла манифеста, узнаете в следующих материалах.

Описание возможностей компонентов

Как было сказано выше, вы можете использовать объект типа Intent для запуска явлений, сервисов и широковещательных приемников.

Вы можете сделать это явно указав имя компонента в намерении (используя имя класса компонента).

Однако, реальная мощь намерений скрывается в концепции неявных намерений.

Неявные намерения просто описывают тип действия для выполнения (и, возможно, необходимые для действия данные), а система автоматически ищет компонент, который может выполнить действие с заданным типом, и запускает его.

Если существует несколько компонентов, способных выполнить действие, пользователь сможет выбрать желаемый компонент из списка.

Поиск компонентов происходит следующим образом: система получает намерение и сверяет его с фильтрами намерений, описанных в манифесте каждого из приложений на устройстве.

Если вы описали явление в манифесте, вы можете по желанию добавить к нему фильтры намерений, которые говорят о возможностях явления, то есть какие намерения это явление может обрабатывать.

Чтобы описать фильтр, необходимо добавить к описанию компонента дочерний элемент <intent-filter>.

Например, если вы создали почтовый клиент, который включает явление для создания нового письма, вы можете описать фильтр для ответа на намерения с действием “send” (чтобы отправить новое письмо), например так:

```
1 <manifest ... >
2 ...
3   <application ... >
4     <activity android:name="com.example.project.ComposeEmailActivity">
5       <intent-filter>
6         <action android:name="android.intent.action.SEND" />
7         <data android:type="*/*" />
8         <category android:name="android.intent.category.DEFAULT" />
9       </intent-filter>
10      </activity>
11    </application>
12 </manifest>
```

Теперь если другое приложение создаст намерение с действием ACTION_SEND и передаст его в метод `startActivity()`, система может запустить ваше явление, чтобы пользователь написал и отправил письмо

Описание требований приложения

Существует много различных устройств на Android, и не у всех из них одинаковые возможности.

Для того, чтобы предотвратить установку приложения на устройства, которые не поддерживают необходимые для работы приложения возможности, важно указать аппаратные и программные зависимости в файле манифеста.

Многие записи являются информационными и система их не считывает, но внешний сервис Google Play использует их и выдает пользователям только совместимые с их устройствами приложения.

Например, если вашему приложению требуется камера и Android не ниже 2.1 (API 7), вы должны указать в манифесте что-то подобное:

```
1 <manifest ... >
2   <uses-feature android:name="android.hardware.camera.any"
3     android:required="true" />
4   <uses-sdk android:minSdkVersion="7" android:targetSdkVersion="19" />
5 ...
6 </manifest>
```

Теперь ваше приложение не сможет быть установлено из Google Play на устройства, в которых нет камеры или версия Android ниже 2.1.

Однако, вы можете также сообщить, что ваше приложение использует камеру, но ее наличие не является обязательным.
В таком случае, установите атрибуте required в значение false и проверяйте наличие камеры в программном коде.

Ресурсы приложения

Android приложения состоят не только из кода – им требуются ресурсы, которые отделены от программного кода, например изображения, звуковые и другие файлы, которые влияют на внешний вид приложения.

К примеру вам потребуется создавать анимацию, меню, стили, цвета и разметку явлений с помощью XML файлов.

Использование ресурсов в приложении делает простым обновление различных характеристик без модификации исходного кода, для этого просто нужно загрузить альтернативные ресурсы, и позволяет оптимизировать приложение под различные конфигурации устройств (например разные языки или размеры экранов).

Каждому ресурсу вашего проекта, SDK присваивает уникальный целочисленный идентификатор, которые вы можете использовать для ссылки на ресурсы из программного кода или из XML файлов других ресурсов.

Например, если ваше приложение содержит изображение с именем logo.png (которое хранится в каталоге res/drawable/), SDK сгенерирует уникальный идентификатор с именем R.drawable.logo, который вы можете использовать для ссылки на изображение и вставки этого изображения в пользовательский интерфейс.

Одним из наиболее важных аспектов отделения ресурсов от кода является возможность создавать различные ресурсы для различных конфигураций устройств. Например, вы можете создать строковые константы для разных языков и хранить их в различных файлах.

Основываясь на спецификаторе языка, который добавляется к названию директорий (например res/value-fr/ для французского) и настройках языка на устройстве, система сама выберет подходящий файл для пользовательского интерфейса.

Спецификатор – это короткая строка, которая добавляется к имени директории, чтобы указать конфигурацию устройства, для которого ресурсы из данной директории будут использоваться.

Android поддерживает множество различных спецификаторов для создания альтернативных ресурсов.

Например, если устройство находится в портретной ориентации, вы можете захотеть выравнять кнопки вертикально, если в ландшафтной – горизонтально.

Чтобы менять макет в зависимости от ориентации экрана, необходимо создать два варианта разметки и применить спецификатор к директориям, в которых разметка будет храниться. Система автоматически будет выбирать нужную.

Совместимость устройств

Android разработан для запуска на множестве различных устройств, начиная от телефонов и заканчивая планшетами и телевизорами.

Для разработчика приложений такое разнообразие устройств предоставляет огромную потенциальную аудиторию пользователей.

Для того, чтобы ваше приложение одинаково хорошо работало на всех устройствах, оно должно учитывать наличие функций на устройствах и предоставлять гибкий пользовательский интерфейс, адаптированный под различные размеры экранов.

Для облегчения ваших усилий при достижении цели, Android предоставляет динамический API, в котором вы можете создавать индивидуальные ресурсы для различных конфигураций устройств (например различную XML разметку для разных размеров экрана).

Android автоматически загружает нужную разметку, основываясь на конфигурации устройства, на котором приложение будет запущено.

Ресурсы для всех вариантов конфигурации могут быть упакованы в один единственный APK файл.

Если это необходимо, вы можете указать требования к аппаратному обеспечению устройства, чтобы приложение не могло быть установлено из Google Play на устройства, которые этим требованиям не соответствуют.

В текущем разделе мы расскажем, как управлять доступностью приложения для разных устройств, чтобы быть уверенным, что приложение достигает нужной аудитории.

Во время изучения системы Android, вы, вероятно, еще не раз столкнетесь со словом “Совместимость” в различных ситуациях.

Есть два типа совместимости: совместимость устройств и совместимость приложений.

Поскольку Android это проект с открытым кодом, любые разработчики железа могут выпускать устройства под операционной системой Android.

До сих пор фраза “устройство совместимо с Android” означало только то, что на нем могут корректно запускаться приложения, написанные для среды выполнения Android.

Каждое устройство должно пройти тест на совместимость CTS (Compatibility Test Suite), для того, чтобы считаться совместимым.

Вам, как разработчику приложений, не стоит беспокоиться о совместимости устройств, поскольку доступ к сервису Google Play Store имеют только совместимые с Android устройства.

Поэтому вы можете быть уверены, что пользователи, установившие ваше приложение из Google Play используют совместимые устройства.

Однако, вы должны учитывать совместимость вашего приложения с различными потенциальными конфигурациями устройств.

Так как устройства под Android имеют широкий спектр конфигураций, не все устройства включают некоторые возможности.

Например, некоторые устройства могут не содержать магнитный сенсор.

Если он требуется вашему приложению для работы, значит ваше приложение совместимо только с устройствами, которые его содержат.

Управление доступностью приложения для устройств

Android поддерживает различные возможности, с которыми ваше приложение может взаимодействовать через API.

Некоторые возможности основаны на аппаратном обеспечении (например магнитный сенсор), другие – на программном обеспечении (например виджеты), есть также возможности, которые доступны только в некоторых версиях платформы Android.

Не каждое устройство поддерживает все функции сразу, поэтому вы должны обеспечить установку приложения только на совместимые устройства.

Чтобы как можно больше пользователей могло пользоваться вашим приложением, вы должны стараться поддерживать как можно больше конфигураций.

В большинстве случаев, вы можете отключить дополнительные функции во время выполнения приложения, если устройство их не поддерживает.

В других случаях вы можете вовсе запретить устанавливать ваше приложение из Google Play, если оно несовместимо по следующим параметрам:

- Возможности устройства
- Версия платформы
- Конфигурация экрана

Возможности устройства

Android определяет идентификаторы для любых аппаратных и программных функций, которые могут быть недоступны на некоторых устройствах. Например, для магнитного сенсора есть идентификатор FEATURE_SENSOR_COMPASS, а для виджетов FEATURE_APP_WIDGETS.

Вы можете запретить устанавливать приложение, если устройство не предоставляет нужных функций, добавив элемент <uses-feature> в файл манифеста.

К примеру, если основная функция вашего приложения требует использование магнитного сенсора, вы должны требовать наличия этого сенсора следующим образом:

```
1 <manifest ... >
2   <uses-feature android:name="android.hardware.sensor.compass"
3     android:required="true" />
4 ...
5 </manifest>
```

Google Play сравнивает функции, которые требует приложение с функциями, доступными на устройстве.

Если устройство не предоставляет все функции, которые вы затребовали, пользователь не сможет установить приложение.

Однако, если для выполнения основных функций приложению не требуется наличие сенсора, установите атрибуте `requires` в значение "false" и проверяйте наличие сенсора программно во время выполнения.

Если сенсор будет недоступен, отключите функции, которые его используют.

Проверить предоставляет ли устройство нужную функцию, можно с помощью метода `hasSystemFeature()`, например так:

```
1 PackageManager pm = getPackageManager();
2 if (!pm.hasSystemFeature(PackageManager.FEATURE_SENSOR_COMPASS)) {
3     // На устройстве нет магнитного сенсора, поэтому отключаем все функции, для
4     // которых он требуется
5     disableCompassFeature();
6 }
```

Примечание: некоторые системные разрешения неявно требуют, чтобы устройство предоставляло функции.

Например, если приложение запрашивает права доступа к Bluetooth, это неявно требует, чтобы bluetooth на устройстве присутствовал (идентификатор FEATURE_BLUETOOTH).

Вы можете сделать приложение доступным для устройств без bluetooth, установив атрибут required в значение "false" в теге <uses-feature>.

Подробности в материале Разрешения, неявно требующие наличие функций.

Версия платформы

На разных устройствах могут быть установлены различные версии Android, например 4.0 или 4.4.

Каждая следующая версия часто добавляет новые возможности в API, недоступные в предыдущих версиях.

Каждая платформа определяет уровень API. Например, Android 1.0 соответствует уровень API 1, а Android 4.4 – уровень API 19.

Уровни API позволяют указать минимальную версию, с которой приложение совместимо.

Для этого служит атрибут `minSdkVersion` тега `<uses-sdk>`.

Например, поставщик содержимого Calendar был добавлен в Android 4.0 (уровень API 14).

Если ваше приложение не может работать без данного API, нужно указать минимальную версию в манифесте:

```
1 <manifest ... >
2   <uses-sdk android:minSdkVersion="14" android:targetSdkVersion="19" />
3 ...
4 </manifest>
```

Атрибут `minSdkVersion` описывает минимальную версию, необходимую для работы приложения.

Другой атрибут, `targetSdkVersion` описывает самую большую версию, под которую оптимизировано приложение.

Каждая последующая версия Android имеет обратную совместимость с приложениями, написанными для предыдущих версий API, поэтому ваше приложение всегда должно быть совместимо с максимально доступной версией.

Примечание: атрибут `targetSdkVersion` не мешает установить приложение на устройства с более высокой версией, но он важен, поскольку указывает системе, должно ли приложение наследовать изменение поведения в новых версиях.

Если вы не обновите атрибут `targetSdkVersion` до последней версии, система будет работать с приложением в режиме обратной совместимости.

Например, в Android 4.4, сигналы, созданные с помощью `AlarmManager` будут объединяться в группы, для экономии энергии, но если `targetSdkVersion` будет меньше 19, система будет сохранять прежнее поведение для данной функции.

Однако, если ваше приложение использует API, которое было добавлено в более поздних версиях, для не особо важных дополнительных функций, необходимо проверять версию API в программном коде во время выполнения и отключать недоступные функции, если уровень API ниже.

В данном случае, установите `minSdkVersion` в минимально возможное значение, которое требуется для работы основных функций приложения и сравнивайте текущую версию системы (константа `SDK_INT`) с одной из констант `Build.VERSION_CODES`, в которых хранятся уровни API.

Например:

```
1 if (Build.VERSION.SDK_INT < Build.VERSION_CODES.HONEYCOMB) {  
2     // Приложение запущено на устройстве с версией API меньше 11, запретим  
3     // функции drag/drop, которые используют ClipboardManager.  
4     disableDragAndDrop();  
5 }
```

Конфигурация экрана

Android устанавливается на устройства с различными размерами экрана, начиная от смартфонов, заканчивая телевизорами.

Для группировки устройств по типу экрана, Android использует две характеристики: размер экрана(физический размер) и плотность пикселей (физическaя плотность пикселей, известная как DPI (точек на дюйм)) и обобщает эти характеристики в группы:

- Четыре размера: маленький (small), нормальный (normal), большой (large) и очень большой (xlarge).
- И несколько вариантов плотности: mdpi (средняя), hdpi (высокая), xhdpi (очень высокая), xxhdpi(ультра высокая) и другие.

По умолчанию ваше приложение совместимо со всеми видами экранов, поскольку система вносит соответствующие изменения в макеты и графические ресурсы по мере необходимости для каждого экрана.

Однако, чтобы это выглядело лучшим образом, вы должны оптимизировать пользовательский интерфейс для каждой конфигурации, добавив индивидуальную разметку и оптимизированные изображения.

Не технические причины доступности приложения

В дополнение к ограничению установки приложения на несовместимые устройства, вы, возможно, захотите ограничить доступность приложения для бизнеса по юридическим причинам.

Или, к примеру, приложение, которое показывает расписание поездов Лондонской подземки, вряд ли полезно для пользователей за пределами Соединенного Королевства.

В таких ситуациях, Google Play предоставляет фильтры в панели разработчика, которые позволяют контролировать доступность приложения по не техническим причинам, таким как локализация устройства или оператор мобильной связи.

Фильтры для технической совместимости всегда располагаются в APK файле. В то время, как фильтры для не технических причин, всегда создаются в панели разработчика Google Play.

Системные разрешения

Android это система с разделением прав, в которой каждое приложение запущено с индивидуальным идентификатором (ID пользователя и ID группы системы Linux).

Части системы также содержат собственные идентификаторы.

Таким образом, Linux изолирует приложения друг от друга и от системы.

Дополнительные функции безопасности предоставляют механизм “разрешений”, который усиливает ограничения на доступ к операциям, которые определенный процесс может выполнять.

Также выдаются разрешения для каждого URI для предоставления индивидуального доступа к определенным частям данных.

В этой лекции мы рассмотрим как разработчики приложений могут использовать функции безопасности, которые предоставляются системой Android

Архитектура безопасности

Центральным звеном архитектуры безопасности Android является то, что по умолчанию, ни одно из приложений не имеет разрешения на выполнение каких-либо операций, которые могут отрицательно повлиять на другие приложения, операционную систему или пользователя.

Эти ограничения включают в себя право чтения или записи личных данных (например контактов или email сообщений), чтение или запись файлов других приложений, доступ к сети, право выключать устройство и так далее.

Поскольку все приложения выполняется в песочнице, они должны явно указывать данные, доступные другим.

Это делается с помощью разрешения дополнительных возможностей, не предусмотренных основной песочницей.

Приложения статически запрашивают разрешения, которые им необходимы, а система Android запрашивает у пользователя согласие во время установки.

В Android нет механизма для пересмотра разрешений динамически во время выполнения программы, поскольку это усложняет работу пользователя в ущерб безопасности.

Изолированная среда приложения не зависит от технологии, используемой при сборке.

В частности, виртуальная машина Dalvik не является преградой для выполнения приложением нативного кода (смотрите Android NDK).

Все типы приложений – Java, нативные и гибридные – запускаются в одинаковых песочницах и защищены одинаково

Подпись приложения

Все APK файлы должны быть подписаны сертификатом, закрытый ключ которого хранится у их разработчика.

Данный сертификат идентифицирует автора приложения.

Сертификаты не обязательно должны быть подписаны центром сертификации; это совершенно допустимо и характерно для Android приложений – использовать собственные сертификаты.

Они позволяют системе предоставлять или отказывать в доступе приложениям с подписанным уровнем доступа, а также разрешать или отказывать приложениям иметь такой-же идентификатор, как у другого приложения.

Идентификаторы пользователя и доступ к файлам

Во время установки, Android назначает каждому пакету уникальный идентификатор пользователя Linux (UID). Идентификатор остается постоянным на всё время существования пакета на этом устройстве.

На другом устройстве тот же пакет может иметь другой UID; важно, чтобы каждый пакет на данном устройстве имел свой собственный идентификатор пользователя.

Поскольку управление безопасностью происходит на уровне процесса, код любых двух пакетов не может нормально работать в одном и том же процессе, так как они должны выполняться под различными пользователями Linux.

Вы можете использовать атрибут `sharedUserId` в файлах манифеста для нескольких пакетов, чтобы указать общий UID. После этого в целях безопасности, два пакета будут выполняться как одно и тоже приложение, с одинаковым UID и одинаковыми правами на файлы.

Помните, что в целях безопасности, только два приложения с одинаковой подписью (и хранящие одинаковый атрибут `sharedUserId`) будут иметь одинаковый идентификатор пользователя.

Любые данные, которые сохраняет приложение, будут связаны с идентификатором приложения и в нормальных условиях будут недоступны другим пакетам.

При создании нового файла с использованием методов `getSharedPreferences(String, int)`, `openFileOutput(String, int)` или `openOrCreateDatabase(String, int, SQLiteDatabase.CursorFactory)`, вы можете использовать флаги `MODE_WORLD_READABLE` и `MODE_WORLD_WRITEABLE`, чтобы разрешить другим пакетам читать или записывать данный файл.

При установке данных флагов, файл все еще принадлежит приложению, но он глобально доступен для чтения и записи любым приложением.

Использование разрешений

Стандартное Android приложение не имеет разрешений, связанных с ним по умолчанию, то есть оно не может сделать ничего, что могло бы негативно повлиять на пользователя или на какие-либо данные.

Для использования защитных функций устройства, вы должны включить в файл `AndroidManifest.xml` один или несколько тегов `<uses-permission>`, указав таким образом разрешения, которые вам нужны.

Например, если приложение должно отслеживать входящие SMS сообщения:

```
1 <manifest xmlns:android="schemas.android.com/apk/res/android"
2   package="com.android.app.myapp" >
3     <uses-permission android:name="android.permission.RECEIVE_SMS" />
4 ...
5 </manifest>
```

Во время установки приложения, запрошенные разрешения предоставляются на основе проверок подписи приложения и взаимодействия с пользователем.

Во время выполнения приложения никакие проверки разрешений с уведомлением пользователя не проводятся; если приложению выдано конкретное разрешение на этапе установки, оно может его использовать по желанию в любое время.

Если разрешение не выдано, любая попытка использовать функцию отменяется без уведомления пользователя.

Часто при отказе в доступе выдается исключение `SecurityException`.

Однако, это не всегда так. Например, метод `sendBroadcast(Intent)` проверяет разрешение на данные, которые передаются каждому приемнику, после чего метод возвращает управление, поэтому вы не получите исключение, если произойдет отказ в доступе. Почти во всех случаях о нарушении разрешений будет сделана запись в системный журнал.

Однако, в нормальной ситуации (например при установке приложения из Google Play), приложение не сможет быть установлено, если пользователь не подтвердил каждое запрошенное разрешение.

В большинстве случаев вам не надо беспокоиться о ошибках времени выполнения, связанных с недостающими разрешениями, поскольку сам факт того, что приложение установлено, означает, что все желаемые разрешения ему были предоставлены.

Разрешения, предоставляемые системой Android, расположены в классе `Manifest.permission`.

Любые приложения могут описывать и требовать свои собственные разрешения, так что это не полный перечень всех возможных разрешений.

Особые разрешения могут быть выполнены в нескольких местах во время работы приложения:

- Во время вызова в системе, чтобы предотвратить выполнение приложением некоторых функций.
- При запуске явления, чтобы избежать запуска приложения другими приложениями.
- Во время отправки и получения широковещательных сообщений, чтобы контролировать кто может получать ваши сообщения и кому вы их можете посыпать.
- Во время доступа и работы с поставщиками содержимого.
- При связывании или запуске сервиса.

Внимание: со временем, в API могут быть добавлены новые ограничения, так что для их использования может понадобиться запрос разрешений, даже если раньше они не требовались.

Поскольку существующие приложения считают доступ к некоторым функциям открытым, Android может добавлять новые разрешения в манифест, чтобы предотвратить нарушение работы приложения в новой версии платформы.

Android принимает решение о необходимости добавить разрешение, основываясь на значении атрибута `targetSdkVersion`.

Если значение ниже, чем версия, в которой разрешение было добавлено, Android сам добавит разрешение в манифест.

Например, разрешение WRITE_EXTERNAL_STORAGE было добавлено в API 4, чтобы закрыть доступ к общему хранилищу.

Если вы установите targetSdkVersion равное 3 или ниже, на новых версиях Android это разрешение будет автоматически добавлено приложению.

Помните, что если такое случится с вашим приложением, на Google Play данное разрешение будет в списке необходимых, хотя ваше приложение может вообще в них не нуждаться.

Чтобы предотвратить такое поведение и удалить разрешения, в которых приложение не нуждается, всегда устанавливайте targetSdkVersion в максимально возможное значение.

Какие разрешения добавлялись в каждой из версий, вы можете посмотреть в материалах Build.VERSION_CODES.

Объявление и требование разрешений

Чтобы требовать собственное разрешение, вы должны сначала объявить его в файле манифеста, используя элемент <permission>.

Например, приложение должно контролировать, кто может запустить одно из его явлений. Для этого объявим новое разрешение следующим образом:

```
1 <manifest xmlns:android="schemas.android.com/apk/res/android"  
2   package="com.me.app.myapp" >  
3   <permission android:name="com.me.app.myapp.permission.DEADLY_ACTIVITY"  
4     android:label="@string/permlab_deadlyActivity"  
5     android:description="@string/permdesc_deadlyActivity"  
6     android:permissionGroup="android.permission-group.COST MONEY"  
7     android:protectionLevel="dangerous" />  
8   ...  
9 </manifest>
```

Обязательный атрибут `<protectionLevel>` сообщает системе, как пользователь будет проинформирован о приложениях, требующих разрешение, или кто имеет право использовать такое разрешение, как описано в связанном документе.

Необязательный атрибут `<permissionGroup>` используется, чтобы помочь системе показать разрешение пользователю. Как правило выбирается либо стандартная системная группа (список групп смотрите здесь `android.Manifest.permission_group`), либо, в более редких случаях, определенная самостоятельно.

Предпочтительно использовать существующую группу, так как это упрощает интерфейс пользователя для демонстрации.

Помните, что метка и описание должны быть заданы.

Это строковые ресурсы, которые могут быть показаны пользователю при просмотре списка разрешений (`android:label`) или при просмотре описания отдельно взятого разрешения (`android:description`).

Метка должна быть короткой, несколько ключевых слов о функциях, которые защищает разрешение.

Описание должно состоять из нескольких предложений, описывающих то, что разрешение позволяет делать.

Мы рекомендуем составить описание из двух частей: в первой описать разрешение, а во второй предупредить пользователя о возможных плохих последствиях

Вот пример описания разрешения CALL_PHONE:

```
<string name="permlab_callPhone">Прямой вызов телефонных  
номеров</string>
```

```
<string name="permdesc_callPhone">Позволяет приложению звонить на  
телефонные номера без вашего вмешательства. Вредоносные программы  
могут использовать эту функцию, чтобы звонить на платные номера за ваш  
счет! Обратите внимание, что это разрешение не позволяет делать звонки на  
номера экстренных служб.</string>
```

Вы можете посмотреть список текущих разрешений в приложении Настройки, а также с помощью команды adb shell pm list permissions.

Для использования приложения Настройки, перейдите в Настройки->Приложения.

Выберите приложение и пролистайте вниз до строки “разрешения”.

Для разработчиков ключ ‘-s’ команды adb позволяет показать разрешения в той форме, в которой его обычно видит пользователь:

```
1 $ adb shell pm list permissions -s
2 <strong>All Permissions</strong>:
3
4 Network communication: view Wi-Fi state, create Bluetooth connections, full
5 Internet access, view network state
6
7 Your location: access extra location provider commands, fine (GPS) location,
8 mock location sources for testing, coarse (network-based) location
9
10 Services that cost you money: send SMS messages, directly call phone numbers
11
12 ...
```

Требование разрешений в AndroidManifest.xml

Разрешения на высоком уровне, ограничивающие доступ ко всем компонентам системы или отдельным приложениям, могут быть установлены в файле AndroidManifest.xml.

Все, что требуется, это добавить атрибут android:permission желаемому компоненту, указав разрешение, которое будет использоваться для контроля доступа к нему.

Разрешения для явлений (применяются к тегу <activity>) указывает, кто может запустить данное явление.

Разрешение проверяется при выполнении методов Context.startActivity() и Activity.startActivityForResult(); если вызывающий компонент не имеет нужного разрешения, будет выброшено исключение SecurityException.

Разрешения для сервисов (применяются к тегу <service>) указывают, кто может запускать или связываться с сервисом.

Разрешения проверяются во время выполнения методов Context.startService(), Context.stopService() и Context.bindService(); если вызывающий компонент не имеет нужного разрешения, будет выброшено исключение SecurityException.

Разрешения для широковещательных приемников (применяются к тегу <receiver>) указывают, кто может отправлять сообщения данному получателю.

Разрешения проверяются после завершения работы метода Context.sendBroadcast(), когда система пытается доставить сообщение данному приемнику.

В результате отказ не приведет к выбросу исключения; просто сообщение не будет доставлено.

В таких случаях, разрешение может быть выставлено в методе Context.registerReceiver(), чтобы контролировать кто может передавать сообщения программно зарегистрированным приемникам.

Можно пойти другим путем, установив разрешения при вызове метода Context.sendBroadcast(), чтобы указать, какие объекты BroadcastReceiver могут получать сообщения.

Разрешения для поставщиков содержимого (применяются к тегу <provider>) указывают, кто может получить доступ к данным, которые предоставляет объект ContentProvider.

(Поставщики содержимого имеют важное дополнительное средство защиты – разрешения для URI, которые описаны ниже).

В отличие от других компонентов, есть два отдельных разрешения, которые можно установить: `android:readPermission`, которое указывает кто может читать из поставщика содержимого, и `android:writePermission`, которое указывает кто в него может записывать.

Помните, что если поставщик защищен правами и на чтение и на запись, получение прав только на запись не подразумевает автоматическое получение прав на чтение.

Разрешения проверяются при первом обращении к поставщику содержимого (если у вас нет ни одного из разрешений, будет выброшено исключение `SecurityException`) и во время выполнения его операций.

Для использования метода `ContentResolver.query()` требуются права на чтение; для методов `ContentResolver.insert()`, `ContentResolver.update()` и `ContentResolver.delete()` требуются права на запись.

Во всех случаях при отсутствии разрешений будет выброшено исключение `SecurityException`.

Требование разрешений при отправке широковещательных сообщений

В дополнение к разрешениям, проверяющим, кто может отправлять намерения зарегистрированным широковещательным приемникам (как описано выше), вы можете также задать разрешения при отправке широковещательного сообщения.

Вызов метода `Context.sendBroadcast()` со строкой разрешения, потребует наличия этого разрешения у получающего приложения.

Помните, что и отправители и получатели могут требовать разрешения. В таких случаях должны быть успешно пройдены обе проверки, чтобы сообщение достигло заданной цели.

Требование других разрешений

Произвольные разрешения могут быть затребованы при любом вызове в службе. Это возможно с помощью вызова метода `Context.checkSelfPermission()`.

Вызов этого метода со строкой разрешения вернет целочисленный индикатор, который показывает, было ли выдано разрешение на момент вызова.

Отметим, что такой подход может быть использован только при поступлении вызова от другого процесса, обычно через интерфейс IDL, опубликованный сервисом или переданного другому процессу иным способом.

Есть несколько других способов проверки разрешений.

Если у вас есть `pid` другого процесса, вы можете использовать для проверки разрешений метод `Context.checkSelfPermission(String, int, int)`.

Если у вас есть имя пакета другого приложения, вы можете использовать напрямую метод пакетного менеджера `PackageManager.checkSelfPermission(String, String)`, чтобы выяснить, было ли предоставлено конкретному пакету указанное разрешение.

Разрешения на URI

Стандартной системы проверки разрешений, описанной выше, не достаточно при использовании поставщиков содержимого.

Поставщики могут защитить себя, используя разрешения на чтение и запись, в то время как их прямые клиенты также должны передавать определенные URI другим приложениям для работы с ними.

Типичный пример – это вложение в письмо в почтовом клиенте.

Доступ к письмам должен быть защищен разрешениями, так как это конфиденциальные данные.

Однако, если URI вложенного изображения передается в просмотрщик изображений, последний не сможет его открыть, поскольку нет причин давать ему доступ ко всей почте.

Решением проблемы являются разрешения для URI: при запуске явления или возвращении результата в явление,зывающий компонент может установить флаги Intent.FLAG_GRANT_READ_URI_PERMISSION и Intent.FLAG_GRANT_WRITE_URI_PERMISSION. Э

ти флаги дают доступ принимающему явлению к конкретному URI, переданному в намерении, независимо от того, имеет ли он разрешения на доступ к данным поставщика содержимого.

Этот механизм позволяет реализовать общую совместимую модель, в которой действие пользователя (при открытии вложения, выборе контакта из списка, и.т.д) приводит к точечному подтверждению разрешений.

Это позволяет сократить число разрешений, применяя их только для тех приложений, которые непосредственно связаны с работой текущего.

Предоставление точечных URI разрешений, однако, требует некоторого взаимодействия с поставщиком содержимого, который владеет этим URI.

Настоятельно рекомендуется, чтобы поставщики содержимого реализовали этот объект и объявили, что он поддерживает с помощью атрибута android:grantUriPermission или тега <grant-uri-permissions>.

Манифест приложения

Каждое приложение должно иметь файл `AndroidManifest.xml` в корневой директории проекта.

Манифест предоставляет важную информацию о приложении системе Android, которую она должна получить прежде чем сможет запустить любой из компонентов.

Среди прочего, манифест служит для следующих вещей:

- Для указания имен Java пакетов приложения. Имя пакета служит уникальным идентификатором приложения.
- Для описания компонентов приложения - явлений, сервисом, широковещательных приемников и поставщиков содержимого, а также для указания имен классов, которые реализуют каждый из компонентов и публикации их возможностей.

Это описание указывает системе Android, какие есть компоненты и при каких условиях они могут быть запущены.

- Для определения процессов, которые используют компоненты приложения.
- Для указания разрешений, которые должно иметь приложение, чтобы получить доступ к защищенной части API или для взаимодействия с другими приложениями.
- Для описания разрешений, которые требуются другим приложениям для взаимодействия с компонентами данного приложения.
- Для указания списка классов `Instrumentation`, предоставляющих профилирование и другую информацию о работе приложения. Используется только при разработке, при публикации удаляется из манифеста.
- Для указания минимального уровня API, который требуется для приложения.
- Для указания списка библиотек, которые должны быть подключены.

Файл манифеста

Базовая структура файла манифеста и все элементы, которые он включает со всеми его атрибутами описан в соответствующем материале, который мы сегодня подробно рассмотрим.

Прежде чем запустить компонент приложения, система Android узнает, какие компоненты доступны, прочитав файл `AndroidManifest.xml` (файл манифеста).

Вы должны описывать все компоненты вашего приложения в этом файле, который должен находиться в корневой директории проекта.

Кроме описания компонентов приложения, манифест служит также для следующих вещей:

- Указание разрешений, которые требуются приложению, например доступ в интернет или чтение контактов.
- Объявление минимального уровня API, который требуется для работы приложения.
- Объявление программных и аппаратных зависимостей, например камеры, bluetooth или экран с поддержкой нескольких прикосновений.
- Указание библиотек, который должны быть подключены к приложению, например Google Maps Library.
- И другое

Перечень элементов

<action>	<meta-data>
<activity>	<permission>
<activity-alias>	<permission-group>
<application>	<permission-tree>
<category>	<provider>
<data>	<service>
<grant-uri-permission>	<supports-screens>
<instrumentation>	<uses-configuration>
<intent-filter>	<uses-feature>
<manifest>	<uses-library>
	<uses-sdk>

Манифест приложения
Подпись приложения.
Требование разрешений в
AndroidManifest.xml
Архитектура безопасности.
Системные разрешения.

Лекция 11+

Рассмотрим следующие вопросы

Системные разрешения.

Архитектура безопасности.

Подпись приложения.

Идентификаторы пользователя и доступ к файлам. Использование разрешений.

Объявление и требование разрешений.

Требование разрешений в AndroidManifest.xml.

Требование разрешений при отправке широковещательных сообщений.

Требование других разрешений. Разрешения на URI.

Манифест приложения. Элементы и атрибуты.

Системные разрешения

Android это система с разделением прав, в которой каждое приложение запущено с индивидуальным идентификатором (ID пользователя и ID группы системы Linux).

Части системы также содержат собственные идентификаторы.

Таким образом, Linux изолирует приложения друг от друга и от системы.

Дополнительные функции безопасности предоставляют механизм “разрешений”, который усиливает ограничения на доступ к операциям, которые определенный процесс может выполнять.

Также выдаются разрешения для каждого URI для предоставления индивидуального доступа к определенным частям данных.

В этой лекции мы рассмотрим как разработчики приложений могут использовать функции безопасности, которые предоставляются системой Android

Архитектура безопасности

Центральным звеном архитектуры безопасности Android является то, что по умолчанию, ни одно из приложений не имеет разрешения на выполнение каких-либо операций, которые могут отрицательно повлиять на другие приложения, операционную систему или пользователя.

Эти ограничения включают в себя право чтения или записи личных данных (например контактов или email сообщений), чтение или запись файлов других приложений, доступ к сети, право выключать устройство и так далее.

Поскольку все приложения выполняется в песочнице, они должны явно указывать данные, доступные другим.

Это делается с помощью разрешения дополнительных возможностей, не предусмотренных основной песочницей.

Приложения статически запрашивают разрешения, которые им необходимы, а система Android запрашивает у пользователя согласие во время установки.

В Android нет механизма для пересмотра разрешений динамически во время выполнения программы, поскольку это усложняет работу пользователя в ущерб безопасности.

Изолированная среда приложения не зависит от технологии, используемой при сборке.

В частности, виртуальная машина Dalvik не является преградой для выполнения приложением нативного кода (смотрите Android NDK).

Все типы приложений – Java, нативные и гибридные – запускаются в одинаковых песочницах и защищены одинаково

Подпись приложения

Все APK файлы должны быть подписаны сертификатом, закрытый ключ которого хранится у их разработчика.

Данный сертификат идентифицирует автора приложения.

Сертификаты не обязательно должны быть подписаны центром сертификации; это совершенно допустимо и характерно для Android приложений – использовать собственные сертификаты.

Они позволяют системе предоставлять или отказывать в доступе приложениям с подписанным уровнем доступа, а также разрешать или отказывать приложениям иметь такой-же идентификатор, как у другого приложения.

Идентификаторы пользователя и доступ к файлам

Во время установки, Android назначает каждому пакету уникальный идентификатор пользователя Linux (UID). Идентификатор остается постоянным на всё время существования пакета на этом устройстве.

На другом устройстве тот же пакет может иметь другой UID; важно, чтобы каждый пакет на данном устройстве имел свой собственный идентификатор пользователя.

Поскольку управление безопасностью происходит на уровне процесса, код любых двух пакетов не может нормально работать в одном и том же процессе, так как они должны выполняться под различными пользователями Linux.

Вы можете использовать атрибут `sharedUserId` в файлах манифеста для нескольких пакетов, чтобы указать общий UID. После этого в целях безопасности, два пакета будут выполняться как одно и тоже приложение, с одинаковым UID и одинаковыми правами на файлы.

Помните, что в целях безопасности, только два приложения с одинаковой подписью (и хранящие одинаковый атрибут `sharedUserId`) будут иметь одинаковый идентификатор пользователя.

Любые данные, которые сохраняет приложение, будут связаны с идентификатором приложения и в нормальных условиях будут недоступны другим пакетам.

При создании нового файла с использованием методов `getSharedPreferences(String, int)`, `openFileOutput(String, int)` или `openOrCreateDatabase(String, int, SQLiteDatabase.CursorFactory)`, вы можете использовать флаги `MODE_WORLD_READABLE` и `MODE_WORLD_WRITEABLE`, чтобы разрешить другим пакетам читать или записывать данный файл.

При установке данных флагов, файл все еще принадлежит приложению, но он глобально доступен для чтения и записи любым приложением.

Использование разрешений

Стандартное Android приложение не имеет разрешений, связанных с ним по умолчанию, то есть оно не может сделать ничего, что могло бы негативно повлиять на пользователя или на какие-либо данные.

Для использования защитных функций устройства, вы должны включить в файл `AndroidManifest.xml` один или несколько тегов `<uses-permission>`, указав таким образом разрешения, которые вам нужны.

Например, если приложение должно отслеживать входящие SMS сообщения:

```
1 <manifest xmlns:android="schemas.android.com/apk/res/android"
2   package="com.android.app.myapp" >
3     <uses-permission android:name="android.permission.RECEIVE_SMS" />
4 ...
5 </manifest>
```

Во время установки приложения, запрошенные разрешения предоставляются на основе проверок подписи приложения и взаимодействия с пользователем.

Во время выполнения приложения никакие проверки разрешений с уведомлением пользователя не проводятся; если приложению выдано конкретное разрешение на этапе установки, оно может его использовать по желанию в любое время.

Если разрешение не выдано, любая попытка использовать функцию отменяется без уведомления пользователя.

Часто при отказе в доступе выдается исключение `SecurityException`.

Однако, это не всегда так. Например, метод `sendBroadcast(Intent)` проверяет разрешение на данные, которые передаются каждому приемнику, после чего метод возвращает управление, поэтому вы не получите исключение, если произойдет отказ в доступе. Почти во всех случаях о нарушении разрешений будет сделана запись в системный журнал.

Однако, в нормальной ситуации (например при установке приложения из Google Play), приложение не сможет быть установлено, если пользователь не подтвердил каждое запрошенное разрешение.

В большинстве случаев вам не надо беспокоиться о ошибках времени выполнения, связанных с недостающими разрешениями, поскольку сам факт того, что приложение установлено, означает, что все желаемые разрешения ему были предоставлены.

Разрешения, предоставляемые системой Android, расположены в классе `Manifest.permission`.

Любые приложения могут описывать и требовать свои собственные разрешения, так что это не полный перечень всех возможных разрешений.

Особые разрешения могут быть выполнены в нескольких местах во время работы приложения:

- Во время вызова в системе, чтобы предотвратить выполнение приложением некоторых функций.
- При запуске явления, чтобы избежать запуска приложения другими приложениями.
- Во время отправки и получения широковещательных сообщений, чтобы контролировать кто может получать ваши сообщения и кому вы их можете посыпать.
- Во время доступа и работы с поставщиками содержимого.
- При связывании или запуске сервиса.

Внимание: со временем, в API могут быть добавлены новые ограничения, так что для их использования может понадобиться запрос разрешений, даже если раньше они не требовались.

Поскольку существующие приложения считают доступ к некоторым функциям открытым, Android может добавлять новые разрешения в манифест, чтобы предотвратить нарушение работы приложения в новой версии платформы.

Android принимает решение о необходимости добавить разрешение, основываясь на значении атрибута `targetSdkVersion`.

Если значение ниже, чем версия, в которой разрешение было добавлено, Android сам добавит разрешение в манифест.

Например, разрешение WRITE_EXTERNAL_STORAGE было добавлено в API 4, чтобы закрыть доступ к общему хранилищу.

Если вы установите targetSdkVersion равное 3 или ниже, на новых версиях Android это разрешение будет автоматически добавлено приложению.

Помните, что если такое случится с вашим приложением, на Google Play данное разрешение будет в списке необходимых, хотя ваше приложение может вообще в них не нуждаться.

Чтобы предотвратить такое поведение и удалить разрешения, в которых приложение не нуждается, всегда устанавливайте targetSdkVersion в максимально возможное значение.

Какие разрешения добавлялись в каждой из версий, вы можете посмотреть в материалах Build.VERSION_CODES.

Объявление и требование разрешений

Чтобы требовать собственное разрешение, вы должны сначала объявить его в файле манифеста, используя элемент <permission>.

Например, приложение должно контролировать, кто может запустить одно из его явлений. Для этого объявим новое разрешение следующим образом:

```
1 <manifest xmlns:android="schemas.android.com/apk/res/android"  
2   package="com.me.app.myapp" >  
3   <permission android:name="com.me.app.myapp.permission.DEADLY_ACTIVITY"  
4     android:label="@string/permlab_deadlyActivity"  
5     android:description="@string/permdesc_deadlyActivity"  
6     android:permissionGroup="android.permission-group.COST MONEY"  
7     android:protectionLevel="dangerous" />  
8   ...  
9 </manifest>
```

Обязательный атрибут `<protectionLevel>` сообщает системе, как пользователь будет проинформирован о приложениях, требующих разрешение, или кто имеет право использовать такое разрешение, как описано в связанном документе.

Необязательный атрибут `<permissionGroup>` используется, чтобы помочь системе показать разрешение пользователю. Как правило выбирается либо стандартная системная группа (список групп смотрите здесь `android.Manifest.permission_group`), либо, в более редких случаях, определенная самостоятельно.

Предпочтительно использовать существующую группу, так как это упрощает интерфейс пользователя для демонстрации.

Помните, что метка и описание должны быть заданы.

Это строковые ресурсы, которые могут быть показаны пользователю при просмотре списка разрешений (`android:label`) или при просмотре описания отдельно взятого разрешения (`android:description`).

Метка должна быть короткой, несколько ключевых слов о функциях, которые защищает разрешение.

Описание должно состоять из нескольких предложений, описывающих то, что разрешение позволяет делать.

Мы рекомендуем составить описание из двух частей: в первой описать разрешение, а во второй предупредить пользователя о возможных плохих последствиях

Вот пример описания разрешения CALL_PHONE:

```
<string name="permlab_callPhone">Прямой вызов телефонных  
номеров</string>
```

```
<string name="permdesc_callPhone">Позволяет приложению звонить на  
телефонные номера без вашего вмешательства. Вредоносные программы  
могут использовать эту функцию, чтобы звонить на платные номера за ваш  
счет! Обратите внимание, что это разрешение не позволяет делать звонки на  
номера экстренных служб.</string>
```

Вы можете посмотреть список текущих разрешений в приложении Настройки, а также с помощью команды adb shell pm list permissions.

Для использования приложения Настройки, перейдите в Настройки->Приложения.

Выберите приложение и пролистайте вниз до строки “разрешения”.

Для разработчиков ключ ‘-s’ команды adb позволяет показать разрешения в той форме, в которой его обычно видит пользователь:

```
1 $ adb shell pm list permissions -s
2 <strong>All Permissions</strong>:
3
4 Network communication: view Wi-Fi state, create Bluetooth connections, full
5 Internet access, view network state
6
7 Your location: access extra location provider commands, fine (GPS) location,
8 mock location sources for testing, coarse (network-based) location
9
10 Services that cost you money: send SMS messages, directly call phone numbers
11
12 ...
```

Требование разрешений в AndroidManifest.xml

Разрешения на высоком уровне, ограничивающие доступ ко всем компонентам системы или отдельным приложениям, могут быть установлены в файле `AndroidManifest.xml`.

Все, что требуется, это добавить атрибут `android:permission` желаемому компоненту, указав разрешение, которое будет использоваться для контроля доступа к нему.

Разрешения для явлений (применяются к тегу `<activity>`) указывает, кто может запустить данное явление.

Разрешение проверяется при выполнении методов `Context.startActivity()` и `Activity.startActivityForResult()`; если вызывающий компонент не имеет нужного разрешения, будет выброшено исключение `SecurityException`.

Разрешения для сервисов (применяются к тегу `<service>`) указывают, кто может запускать или связываться с сервисом.

Разрешения проверяются во время выполнения методов `Context.startService()`, `Context.stopService()` и `Context.bindService()`; если вызывающий компонент не имеет нужного разрешения, будет выброшено исключение `SecurityException`.

Разрешения для широковещательных приемников (применяются к тегу <receiver>) указывают, кто может отправлять сообщения данному получателю.

Разрешения проверяются после завершения работы метода Context.sendBroadcast(), когда система пытается доставить сообщение данному приемнику.

В результате отказ не приведет к выбросу исключения; просто сообщение не будет доставлено.

В таких случаях, разрешение может быть выставлено в методе Context.registerReceiver(), чтобы контролировать кто может передавать сообщения программно зарегистрированным приемникам.

Можно пойти другим путем, установив разрешения при вызове метода Context.sendBroadcast(), чтобы указать, какие объекты BroadcastReceiver могут получать сообщения.

Разрешения для поставщиков содержимого (применяются к тегу <provider>) указывают, кто может получить доступ к данным, которые предоставляет объект ContentProvider.

(Поставщики содержимого имеют важное дополнительное средство защиты – разрешения для URI, которые описаны ниже).

В отличие от других компонентов, есть два отдельных разрешения, которые можно установить: `android:readPermission`, которое указывает кто может читать из поставщика содержимого, и `android:writePermission`, которое указывает кто в него может записывать.

Помните, что если поставщик защищен правами и на чтение и на запись, получение прав только на запись не подразумевает автоматическое получение прав на чтение.

Разрешения проверяются при первом обращении к поставщику содержимого (если у вас нет ни одного из разрешений, будет выброшено исключение `SecurityException`) и во время выполнения его операций.

Для использования метода `ContentResolver.query()` требуются права на чтение; для методов `ContentResolver.insert()`, `ContentResolver.update()` и `ContentResolver.delete()` требуются права на запись.

Во всех случаях при отсутствии разрешений будет выброшено исключение `SecurityException`.

Требование разрешений при отправке широковещательных сообщений

В дополнение к разрешениям, проверяющим, кто может отправлять намерения зарегистрированным широковещательным приемникам (как описано выше), вы можете также задать разрешения при отправке широковещательного сообщения.

Вызов метода `Context.sendBroadcast()` со строкой разрешения, потребует наличия этого разрешения у получающего приложения.

Помните, что и отправители и получатели могут требовать разрешения. В таких случаях должны быть успешно пройдены обе проверки, чтобы сообщение достигло заданной цели.

Требование других разрешений

Произвольные разрешения могут быть затребованы при любом вызове в службе. Это возможно с помощью вызова метода `Context.checkSelfPermission()`.

Вызов этого метода со строкой разрешения вернет целочисленный индикатор, который показывает, было ли выдано разрешение на момент вызова.

Отметим, что такой подход может быть использован только при поступлении вызова от другого процесса, обычно через интерфейс IDL, опубликованный сервисом или переданного другому процессу иным способом.

Есть несколько других способов проверки разрешений.

Если у вас есть `pid` другого процесса, вы можете использовать для проверки разрешений метод `Context.checkSelfPermission(String, int, int)`.

Если у вас есть имя пакета другого приложения, вы можете использовать напрямую метод пакетного менеджера `PackageManager.checkSelfPermission(String, String)`, чтобы выяснить, было ли предоставлено конкретному пакету указанное разрешение.

Разрешения на URI

Стандартной системы проверки разрешений, описанной выше, не достаточно при использовании поставщиков содержимого.

Поставщики могут защитить себя, используя разрешения на чтение и запись, в то время как их прямые клиенты также должны передавать определенные URI другим приложениям для работы с ними.

Типичный пример – это вложение в письмо в почтовом клиенте.

Доступ к письмам должен быть защищен разрешениями, так как это конфиденциальные данные.

Однако, если URI вложенного изображения передается в просмотрщик изображений, последний не сможет его открыть, поскольку нет причин давать ему доступ ко всей почте.

Решением проблемы являются разрешения для URI: при запуске явления или возвращении результата в явление,зывающий компонент может установить флаги Intent.FLAG_GRANT_READ_URI_PERMISSION и Intent.FLAG_GRANT_WRITE_URI_PERMISSION. Э

ти флаги дают доступ принимающему явлению к конкретному URI, переданному в намерении, независимо от того, имеет ли он разрешения на доступ к данным поставщика содержимого.

Этот механизм позволяет реализовать общую совместимую модель, в которой действие пользователя (при открытии вложения, выборе контакта из списка, и.т.д) приводит к точечному подтверждению разрешений.

Это позволяет сократить число разрешений, применяя их только для тех приложений, которые непосредственно связаны с работой текущего.

Предоставление точечных URI разрешений, однако, требует некоторого взаимодействия с поставщиком содержимого, который владеет этим URI.

Настоятельно рекомендуется, чтобы поставщики содержимого реализовали этот объект и объявили, что он поддерживает с помощью атрибута android:grantUriPermission или тега <grant-uri-permissions>.

Манифест приложения

Каждое приложение должно иметь файл `AndroidManifest.xml` в корневой директории проекта.

Манифест предоставляет важную информацию о приложении системе Android, которую она должна получить прежде чем сможет запустить любой из компонентов.

Среди прочего, манифест служит для реализации следующего функционала:

- Для указания имен Java пакетов приложения. Имя пакета служит уникальным идентификатором приложения.
- Для описания компонентов приложения - явлений, сервисов, широковещательных приемников и поставщиков содержимого, а также для указания имен классов, которые реализуют каждый из компонентов и публикации их возможностей.

Это описание указывает системе Android, какие есть компоненты и при каких условиях они могут быть запущены.

- Для определения процессов, которые используют компоненты приложения.
- Для указания разрешений, которые должно иметь приложение, чтобы получить доступ к защищенной части API или для взаимодействия с другими приложениями.
- Для описания разрешений, которые требуются другим приложениям для взаимодействия с компонентами данного приложения.
- Для указания списка классов `Instrumentation`, предоставляющих профилирование и другую информацию о работе приложения. Используется только при разработке, при публикации удаляется из манифеста.
- Для указания минимального уровня API, который требуется для приложения.
- Для указания списка библиотек, которые должны быть подключены.
- Объявление программных и аппаратных зависимостей, например камеры, bluetooth или экран с поддержкой нескольких прикосновений.

Файл манифеста

Базовая структура файла манифеста и все элементы, которые он включает со всеми его атрибутами описан в соответствующем материале, который мы сегодня начнем подробно рассматривать.

Напомню, что первичной задачей файла манифеста является информирование системы о компонентах приложения.

Прежде чем запустить компоненты приложения, система Android узнает, какие компоненты доступны, прочитав файл `AndroidManifest.xml` (файл манифеста).

Вы должны описывать все компоненты вашего приложения в этом файле, который должен находиться в корневой директории проекта.

Перечень элементов манифеста

<action>	<meta-data>
<activity>	<permission>
<activity-alias>	<permission-group>
<application>	<permission-tree>
<category>	<provider>
<data>	<service>
<grant-uri-permission>	<supports-screens>
<instrumentation>	<uses-configuration>
<intent-filter>	<uses-feature>
<manifest>	<uses-library>
	<uses-sdk>

Все основные доступные элементы рассмотрим далее. Вы не можете добавлять в манифест свои собственные элементы..

Язык XML как способ представления объектной информации

Зная свойства объектов и основные их характеристики, возникает закономерный вопрос: как представить это описание так, чтобы оно было понятно как человеку, так и компьютеру?

Для этого можно использовать различные подходы, одним из которых является расширяемый язык разметки электронных XML-документов.

XML («eXtensible Markup Language») — это расширяемый язык разметки, рекомендованный консорциумом Всемирной паутины («W3C») для описания электронных документов.

Современные языки разметки документов прошли долгий путь от первых форм, создававшихся компаниями и госучреждениями, до стандартного языка обобщенной разметки SGML (Standard Generalized Markup Language), гипертекстового языка разметки HTML (Hypertext Markup Language) и, в конечном итоге, до XML.

Язык SGML является достаточно сложным для восприятия человеком, а язык HTML оказался недостаточно мощным для обработки компьютерами.

В этой ситуации язык XML стал некоторым компромиссом. Он является одновременно достаточно простым для восприятия человеком и удобным для обработки компьютером.

Слогической точки зрения XML-документ состоит из двух частей:

1. Пролог — это необязательная часть, предназначенная для хранения дополнительной или служебной информации о самом XML-документе.
2. Корневой элемент — это обязательная часть документа, составляющая его содержание.

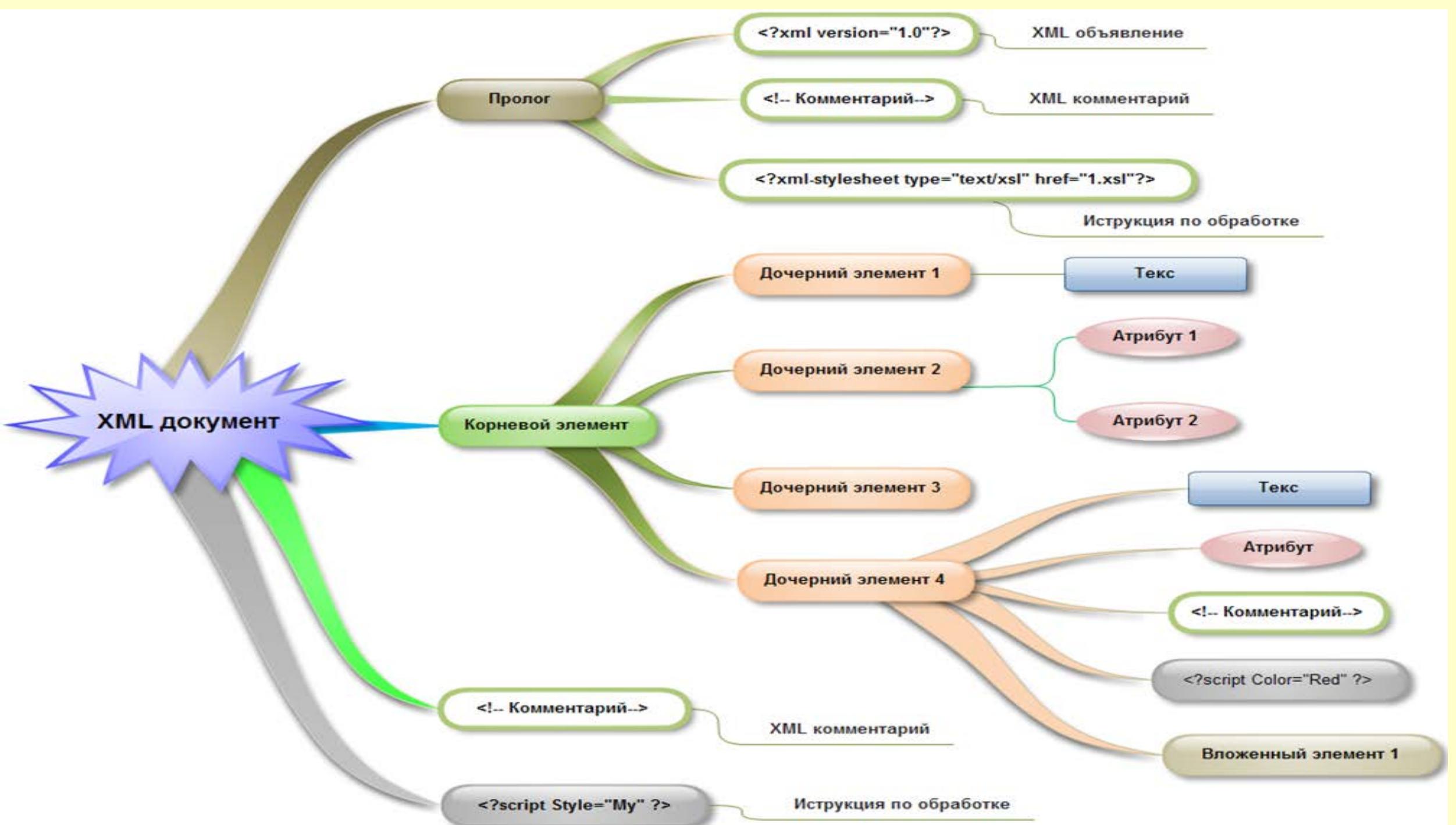
Пролог может включать в себя:

1. XML-декларацию.
2. XML-комментарии.
3. Инструкции по обработке.

Корневой элемент в документе может быть только одним. Однако он может иметь атрибуты и включать в себя:

1. Символьные данные.
2. Вложенные элементы.
3. XML-комментарии.
4. Инструкции по обработке.

Комментарии и инструкции по обработке могут находиться в любом месте документа, перед корневым элементом, внутри него и даже после него



Элемент манифеста <action>

СИНТАКСИС:

```
<action android:name="string" />
```

СОДЕРЖИТСЯ В:

```
<intent-filter>
```

ОПИСАНИЕ:

Добавляет действие для фильтра намерения. Элемент `<intent-filter>` должен содержать один или несколько элементов `<action>`, иначе ни одно намерение не сможет пройти через данный фильтр

АТРИБУТЫ:

`android: name`

Наименование действия.

Некоторые стандартные действия объявлены в классе Intent как константы ACTION_ строка.

Чтобы назначить одно из этих действий атрибуту, добавьте значение "android.intent.action" перед строкой, которая следует за ACTION_.

Например, для действия ACTION_MAIN, используйте "android.intent.action.MAIN", а для действия ACTION_WEB_SEARCH, соответственно "android.intent.action.WEB_SEARCH".

Для действий, которые создали вы, лучше всего использовать в качестве префикса имя пакета, чтобы гарантировать уникальность.

Например, действие TRANSMOGRIFY может быть объявлено следующим образом:

```
<action android:name="com.example.project.TRANSMOGRIFY" />
```

ДОБАВЛЕНО: уровень API 1

Элемент манифеста <activity>

СИНТАКСИС:

```
<activity android:allowEmbedded=["true" | "false"]
          android:allowTaskReparenting=["true" | "false"]
          android:alwaysRetainTaskState=["true" | "false"]
          android:autoRemoveFromRecents=["true" | "false"]
          android:banner="drawable resource"
          android:clearTaskOnLaunch=["true" | "false"]
          android:configChanges=["mcc", "mnc", "locale",
                                 "touchscreen", "keyboard", "keyboardHidden",
                                 "navigation", "screenLayout", "fontScale", "uiMode",
                                 "orientation", "screenSize", "smallestScreenSize"]
          android:documentLaunchMode=["intoExisting", "always",
                                      "none", "never"]
```

android:enabled=["true" | "false"]
 android:excludeFromRecents=["true" | "false"]
 android:exported=["true" | "false"]
 android:finishOnTaskLaunch=["true" | "false"]
 android:hardwareAccelerated=["true" | "false"]
 android:icon="drawable resource"
 android:label="string resource"
 android:launchMode=["multiple" | "singleTop" |
 "singleTask" | "singleInstance"]

```
    android:maxRecents="integer"
    android:multiprocess=["true" | "false"]
    android:name="string"
    android:noHistory=["true" | "false"]
    android:parentActivityName="string"
    android:permission="string"
    android:process="string"
    android:relinquishTaskIdentity=["true" | "false"]
    android:screenOrientation=["unspecified" | "behind" |
        "landscape" | "portrait" |
        "reverseLandscape" | "reversePortrait" |
        "sensorLandscape" | "sensorPortrait" |
```

```
"userLandscape" | "userPortrait" |
    "sensor" | "fullSensor" | "nosensor" |
    "user" | "fullUser" | "locked"]
    android:stateNotNeeded=["true" | "false"]
    android:taskAffinity="string"
    android:theme="resource or theme"
    android:uiOptions=["none" | "splitActionBarWhenNarrow"]
    android:windowSoftInputMode=["stateUnspecified",
        "stateUnchanged", "stateHidden",
        "stateAlwaysHidden", "stateVisible",
        "stateAlwaysVisible", "adjustUnspecified",
        "adjustResize", "adjustPan"] >
    ...
</activity>
```

СОДЕРЖИТСЯ В:

<application>

МОЖЕТ СОДЕРЖАТЬ

<intent-filter>

<meta-data>

ОПИСАНИЕ:

Объявляет явление (подкласс Activity), которое реализует часть пользовательского интерфейса.

Для каждого явления должен существовать элемент <activity> в манифесте.

Если явление не объявлено в манифесте, оно недоступно для системы и никогда не сможет быть запущено

АТРИБУТЫ:

android:allowEmbedded

Показывает, что явление может быть запущено как встроенный дочерний элемент другого явления.

В частности в случае, когда дочерний элемент находится в контейнере, вроде `Display`, принадлежащем другому явлению.

Например, явление, которое используется для уведомлений на носимых устройствах, должно объявлять что такое устройство может отобразить явление в контексте его потока, который находится в другом процессе.

Значение по умолчанию `false`

android:allowTaskReparenting

Указывает может ли явление перемещаться из задачи, которая его запустила в родственную задачу, когда эта задача будет выведена на передний план: "true" если может перемещаться, "false" если должно оставаться в задаче, в которой было запущено.

Если атрибут не установлен, будет применен атрибут `allowTaskReparenting` элемента `<application>`. Значение по умолчанию "false"

Обычно явление остается связанным с задачей, которая его запустила.

Вы можете использовать данный атрибут, чтобы заставить явление повторно использоваться в задаче-потомке, когда родительская задача больше не отображается.

Как правило, он используется чтобы вызвать явление и переместить его в главную задачу, ассоцииированную с приложением.

Например, e-mail сообщение содержит ссылку на веб-страницу.

Щелчок по ссылке откроет явление, способное отобразить веб-страницу.

Это явление объявлено в приложении-браузере, но открывается как часть почтового клиента.

Если переместить явление в задачу браузера, а затем открыть браузер, явление будет находиться уже внутри браузера.

При этом, если вернуться в почтовый клиент, этого явления там уже не будет.

Родство явлений объявляется с помощью атрибута `taskAffinity`.

Родство задачи определяется путем получения родства корневого явления.

Таким образом, по определению, корневое явление всегда выполняется в задаче с таким же родством.

Так явление с режимом запуска "singleTask" или "singleInstance" могут быть только в корневой задаче, смена родителя ограничена режимами "standard" и "singleTop".
(Смотрите также атрибут `launchMode`)

android:alwaysRetainTaskState

Указывает должна ли система удерживать состояние задачи: "true" – должна, "false", если система позволяет сбрасывать задачу в исходное состояние в определенных ситуациях. Значение по умолчанию "false".

Атрибут имеет значение только для корневого явления задачи, для остальных явлений он игнорируется.

Обычно система очищает задачи (удаляет все явления из стека, которые выше корневого явления) в определенных ситуациях, если пользователь заново открыл задачу с домашнего экрана.

Как правило, это будет сделано, если пользователь не посещал задачу в течение определенного времени, например 30 минут.

Однако, если значение атрибута равно "true", пользователь всегда может вернуться к задаче и начать работу с последнего состояния, независимо от того, как он в нее попал.

Это удобно, например, в приложениях вроде веб-браузера, состояние которого (множество открытых вкладок) пользователь не захочет потерять

android:autoRemoveFromRecents

Указывает, должны ли задачи, запущенные из явления с этим атрибутом находиться в списке последних приложений, до тех пор, пока явление не будет завершено.

Если значение "true", задача автоматически удаляется из списка.

Этот атрибут отменяет действие флага FLAG_RETAIN_IN_RECENTS.

Значение должно быть "true" или "false"

android:banner

Ресурс типа `drawable` предоставляет расширенный графический баннер для связанного элемента.

Используется с тегом `<activity>` чтобы установить баннер по умолчанию для конкретного явления, или для тега `<application>`, чтобы установить баннер для всех явлений приложения.

Система использует баннер для представления приложения на домашнем экране Android TV.

Поскольку баннер отображается только на домашнем экране, он должен быть указан только для приложений, явления которых обрабатывают намерение с категорией `CATEGORY_LEANBACK_LAUNCHER`.

Значением атрибута должна быть ссылка на ресурс типа `drawable`, содержащий изображение (например `"@drawable/banner"`).

Баннера по умолчанию нет.

.

android:clearTaskOnLaunch

Указывает, должно ли явление удаляться из задачи, за исключением корневого явления, всякий раз, когда они заново запущены с домашнего экрана: "true" если задача всегда должны начинать с корневого явления, "false" – если нет.

Атрибут имеет значение только для явлений, которые запускают новую задачу (корневые явления), для всех остальных явлений атрибут игнорируется.

Если значение равно true, каждый раз, когда пользователь запускает задачу снова, он перемещается в корневое явление, независимо от того, что он делал в задаче до этого и независимо от того, нажал он кнопку “Назад” или “Домой”, чтобы эту задачу покинуть.

Если значение равно false, задача может быть очищена от явлений (смотрите атрибут alwaysRetainTaskState), но не всегда.

Предположим, например, что кто-то запустил явление P с домашнего экрана и перешел в явление Q.

Затем пользователь нажал кнопку “домой”, а затем вернулся к явлению P. Как правило пользователь увидит явление Q, поскольку это последнее, что он делал в задаче.

Однако, если атрибут явления P будет установлен в true, все явления (Q в данном случае) будут удалены при нажатии кнопки “домой”, когда задача уходит на задний план.

И тогда пользователь увидит явление P при возвращении к задаче.

Если одновременно данный атрибут и атрибут allowTaskReparenting будут установлены в true, все явления, которые могут быть перемещены, будут перемещены в родительские задачи, а остальные явления будут уничтожены, как описано выше

android:configChanges

Список изменений конфигурации, которые явление самостоятельно обрабатывает.

Если конфигурация устройства меняется во время выполнения приложения, система по умолчанию уничтожает и заново создает явление, но указание конфигураций в этом атрибуте предотвращает перезапуск явления.

Вместо этого, явление продолжает работать и вызывает метод `onConfigurationChanged()`.

Примечание: использование этого атрибута следует избегать и применять его только в крайнем случае.

Далее (в таблице) перечислены возможные значения атрибута.

Можно указывать несколько значений, разделяя их вертикальной чертой "|", например "locale|navigation|orientation".

Значение	Описание
mcc	Изменился код страны (IMSI MCC) – SIM карта была обнаружена и обновился код страны.
mnc	Изменился код мобильной сети (IMSI MNC) – SIM карта была обнаружена и обновился код сети.
locale	Изменилась локаль – пользователь выбрал новый язык.
touchscreen	Изменился тачскрин (в нормальной ситуации это никогда не происходит)
keyboard	Изменилась клавиатура – например пользователь подключил внешнюю клавиатуру.
keyboardHidden	Доступность клавиатуры изменилась – например пользователь стал использовать аппаратную клавиатуру
navigation	изменилось устройство навигации (например трекбол). В нормальной ситуации такое никогда не происходит.
screenLayout	изменился экран – такое может происходить при активации другого экрана.
fontScale	Изменился масштаб шрифта – пользователь выбрал новый глобальный размер шрифта.

изменился режим пользовательского интерфейса – такое может uiMode происходить, если пользователь вставил устройство в док-станцию или изменил ночной режим. Смотрите [UiModeManager](#). Добавлено в API 8.

изменилась ориентация экрана – пользователь повернул устройство.

`orientation` **Примечание:** если целевое API равно 13 и выше (задается атрибутами `minSdkVersion` и `targetSdkVersion`), вы должны также указать в атрибуте конфигурацию `screenSize`, поскольку эта конфигурация также меняется при изменении ориентации.

Изменился текущий размер экрана. Сюда же относится изменение пропорций экрана, так что конфигурация меняется при повороте устройства из портретной ориентации в ландшафтную и обратно.

`screenSize` Однако, если вы используете целевое API 12 или ниже, ваше приложение всегда должно обрабатывать изменение этой конфигурации самостоятельно (изменение этой конфигурация не вызывает перезапуска явлений, даже если приложение запущено на Android 3.2 и выше). Добавлено в API 13.

layoutDirection

Изменился физический размер экрана. Этот параметр означает изменение размера экрана вне зависимости от ориентации, поэтому единственный вариант – это переключение вывода изображения на внешний дисплей. Изменение этой `smallestScreenSize` конфигурации соответствует изменению конфигурации [`smallestWidth`](#). Однако, если целевое API равно 12 и выше, явление всегда обрабатывает изменение данной конфигурации самостоятельно (ее изменение не вызывает перезапуск явления даже на устройствах с Android 3.2 и выше). Добавлено в API 13. Изменилось направление разметки, например “слева-на-право” на “справа-на-лево”. Добавлено в API 17.

Все эти изменения конфигурации могут повлиять на значения ресурсов, которые видит приложение. Поэтому при вызове `onConfigurationChanged()` крайне необходимо снова получить все ресурсы (включая разметку, `drawable` и прочие), чтобы корректно обработать изменение

android:documentLaunchMode

Указывает, как новый экземпляр явления должен быть добавлен к задаче при каждом его запуске.

Этот атрибут разрешает пользователям иметь несколько документов приложения, которые будут показаны в списке последних задач.

Атрибут имеет четыре значения, которые производят следующие эффекты при открытии документа в приложении

Примечание: для значений, отличны от none и never, явление должно указывать режим launchMode="standard".

Если данный атрибут не указан, используйте documentLaunchMode="none".

Значение

Описание

introExisting

Явление использует существующую задачу для документа. Данное значение аналогично флагу [FLAG ACTIVITY NEW DOCUMENT](#) без установки флага [FLAG ACTIVITY MULTIPLE TASK](#), как описано в материале [Последние задачи](#).

always

Явление создает новую задачу для документа, даже если документ уже открыт. Это аналогично установке обоих флагов [FLAG ACTIVITY NEW DOCUMENT](#) и [FLAG ACTIVITY MULTIPLE TASK](#).

none

Явление не создает новую задачу для документа. Это значение по умолчанию, при котором новая задача создается только при установленном флаге [FLAG ACTIVITY NEW TASK](#). Список последних задач рассматривает явление, как это было бы по умолчанию: он показывает одну задачу приложения, которая содержит явление, которое пользователь вызвал последним.

never

Явление не будет запущено с новым документом, даже если намерение включает флаг [FLAG ACTIVITY NEW DOCUMENT](#). Установка этого значения затирает значение флагов [FLAG ACTIVITY NEW DOCUMENT](#) и [FLAG ACTIVITY MULTIPLE TASK](#), если какой-либо из них установлен в явлении, список последних задач показывает одну задачу для приложения, которая работает с явлением, запущенным последним.

android:enabled

Указывает, может ли явление быть создано системой. true если может, false – если нет. Значение по умолчанию true.

Элемент<application> имеет свой атрибут enabled, который применяется для всех компонентов приложения, включая явления.

Чтобы система смогла создать явление, атрибуты обоих элементов <application> и <activity> должны быть установлены в true (это их значение по умолчанию).

android:excludeFromRecents

Указывает должна ли задача, запущенная этим явлением, быть исключена из списка последних приложений.

Укажите "true", чтобы не показывать задачу в списке последних приложений. Значение по умолчанию false.

android:exported

Указывает, может ли явление быть запущенным из другого приложения. "true" если может и "false" если нет.

Если установлено значение `false`, явление сможет быть запущено только из его собственного приложения или из приложения с таким же идентификатором пользователя (User ID).

Значение по умолчанию зависит от того, содержит ли явление фильтры намерений.

Отсутствие фильтров подразумевает, что явление может быть вызвано только явно по имени класса, то есть оно существует только для внутреннего пользования (если, конечно, другие не знают его имя).

В таком случае значение по умолчанию "`false`".

С другой стороны, наличие хотя бы одного фильтра означает, что явление предназначено для внешнего применения, и значение по умолчанию равно `true`.

Этот атрибут не единственный способ ограничить действие других приложений на явление.

Вы также должны использовать разрешения для ограничения использования явления внешними объектами

android:finishOnTaskLaunch

Указывает, должен ли существующий экземпляр явления уничтожаться, если пользователь заново запустил эту задачу (с домашнего экрана). true если должен, false если нет.

Значение по умолчанию false.

Если одновременно данный атрибут и атрибут allowTaskReparenting имеют значение true, данный атрибут перекрывает его.

Родственность явлений игнорируется.

Явление не перемещается, а уничтожается

android:hardwareAccelerated

Указывает должно ли использоваться аппаратное ускорение для отрисовки данного явления, true если должна, false если нет.

Значение по умолчанию false.

Начиная с Android 3.0, аппаратное ускорение OpenGL доступно приложениям, для улучшения производительности для большинства распространенных 2D графических операций.

Если аппаратное ускорение разрешено, большинство операций с объектами Canvas, Paint, Xrefmode, ColorFilter, Shader и Camera ускоряются.

В результате получается гладкая анимация, гладкое прокручивание и улучшенная отзывчивость во всем, даже в приложениях, которые явно не используют библиотеку OpenGL.

Поскольку для аппаратного ускорения требуется больше ресурсов, ваше приложение будет потреблять больше памяти.

Помните, что не все операции OpenGL 2D ускоряются.

Если вы разрешили аппаратное ускорение, протестируйте ваше приложение, чтобы убедиться, что все отрисовывается без ошибок.

android:icon

Иконка явления. Отображается пользователю когда требуется показать явление на экране.

Например, иконка явления отображается в списке приложений.

Иконка часто используется одновременно с заголовком явления. (смотрите атрибуте android:label)

Атрибут должен содержать ссылку на ресурс типа `drawable` с изображением.

Если иконка для явления не установлена, будет использована иконка, указанная в атрибуте `android:icon` элемента `<application>`.

Иконка, указанная в этом атрибуте, является также иконкой по умолчанию для всех фильтров намерений (смотрите элемент `<intent-filter>`).

android:label

Заголовок явления, который виден пользователю.

Часто используется вместе с иконкой.

Если атрибут не установлен, будет использоваться значение атрибута label элемента <application>.

Заголовок, указанный в атрибуте, является также заголовком по умолчанию для всех фильтров намерений (смотрите элемент <intent-filter>).

Атрибут должен содержать ссылку на строковый ресурс, это поможет локализовать приложение.

Однако во время разработки временно можно установить в качестве значения простую строку

android:launchMode

Указывает как явление должно быть запущено.

Есть четыре режима работы вместе с флагами (константы FLAG_ACTIVITY_*) в объекте Intent, которые определяют, что должно произойти, когда явление вызвано из намерения:

"standard"

"singleTop"

"singleTask"

"singleInstance"

Режим по умолчанию "standard".

Как показано в таблице ниже, режимы делятся на две группы, "standard" и "singleTop" в одной группе, "singleTask" и "singleInstance" в другой.

Явления с режимами первой группы могут быть создано несколько раз.

Экземпляры могут принадлежать любой задаче и находиться в любом месте стека.

Обычно они запускаются в той же задаче, из которой вызван метод startActivity() (только если намерение не содержит флаг FLAG_ACTIVITY_NEW_TASK, в этом случае будет выбрана другая задача).

Напротив, явления в режимах "singleTask" или "singleInstance" могут только начинать задачу.

Они всегда в корне стека. Кроме того, устройство может содержать только один экземпляр явления и в то же время только одну задачу.

Режимы "standard" и "singleTop" отличаются друг от друга только в одном отношении: каждый раз при отправке нового намерения для явления в режиме "standard", создается новый экземпляр явления для обработки намерения. Каждый экземпляр обрабатывает единственное намерение.

И так же, может быть создан новый экземпляр явления в режиме "singleTop" для обработки намерения.

Однако, если в задаче уже существует экземпляр явления, который находится в вершине стека, он вернет объект Intent в метод onNewIntent(); новый экземпляр не создается.

При других обстоятельствах, например, если существующий экземпляр явления "singleTop" содержится в задаче, но находится не в вершине стека, или находится в вершине стека, но другой задачи – новый экземпляр будет создан и помещен в стек.

Также, если вы вернулись “назад” к родительскому явлению в текущем стеке, поведение зависит от режима запуска родительского явления.

Если оно имеет режим singleTop (или намерение включает флаг FLAG_ACTIVITY_CLEAR_TOP), родитель поместится в вершину стека, и его состояние сохраняется.

Объект Intent, используемый при навигации будет передан в родительское явление в метод onNewIntent().

Если родительское явление имеет режим standard (и намерение не включает флаг FLAG_ACTIVITY_CLEAR_TOP), текущее явление и его родитель будут вытолкнуты из стека, и будет создан новый экземпляр родителя для получения намерения.

Режимы "singleTask" и "singleInstance" тоже имеют всего одно отличие: явление в режиме singleTask разрешает другим явлениям быть частью его задачи.

Оно всегда выполняется в корне задачи, но другие явления (обязательно standard или singleTop) могут быть также запущены в этой задаче.

singleInstance, напротив не допускает, чтобы другие явления были частью его задачи. Это касается только явлений в задаче.

Если запускается другое явление, оно присваивается другой задаче – как если бы был установлен флаг FLAG_ACTIVITY_NEW_TASK в объекте Intent.

Случаи использования	Режим	Несколько экземпляров?	Комментарий
Нормальный запуск для большинства явлений	standard	Да	По умолчанию. Система всегда создает новый экземпляр явления в задаче и направляет намерение в него.
	singleTop	условно	Если экземпляр явления уже существует, в вершине стека задачи, система направляет намерение в него с помощью метода <code>onNewIntent()</code> , а не создает новый экземпляр.
Специализированный запуск (не рекомендуется для общего пользования).	singleTask	Нет	Система создает явление в корне задачи и направляет в него намерение. Однако, если экземпляр явления уже существует, система направит намерение в него с помощью
Специализированный запуск (не рекомендуется для общего пользования).	singleInstance	Нет	Подобно режиму <code>singleTask</code> , за исключением того, что система не запускает другие явления в задаче. Явление всегда одно и является единственным членом задачи

Как показано в таблице выше, standard это режим по умолчанию для большинства типов явлений. SingleTop также распространен и удобен для запуска многих явлений.

Остальные два режима singleTask и singleInstance – не подходят для большинства задач, поскольку они приводят к модели взаимодействия, которая может быть незнакома пользователям и отличается от большинства других приложений.

Несмотря на режим запуска, который вы выберите, проверьте корректность работы явлений во время запуска и при возвращении к нему, в том числе используя кнопку “назад”.

android:maxRecents

Максимальное число задач явления в списке последних задач. Когда число задач достигает указанного значения, система удаляет наиболее старые экземпляры из списка.

Значения могут быть от 1 до 50 (25 для устройств с небольшим количеством памяти, 0 – некорректное значение).

Значение должно быть целочисленным. По умолчанию равно 16.

android:multiprocess

Указывает, может ли экземпляр явления работать в процессе компонента, который его запустил. true если может, false если нет.

По умолчанию false.

Как правило, новый экземпляр явления запускается в процессе приложения, в котором это явление описано, так что все его экземпляры работают в одном и том же процессе.

Однако если атрибут установлен в значение true, экземпляры явлений могут работать в разных процессах, что позволяет системе создавать их везде, где они используются (если есть соответствующее разрешение).

Впрочем, практически не существует случаев, когда такое поведение является необходимым

android:name

Имя класса (подкласс Activity), который реализует данное явление.

Значение атрибута должно содержать полное имя, включая название пакета (например, com.example.project.ExtracurricularActivity).

Тем не менее, в качестве сокращения, если первый символ названия точка (например, .ExtracurricularActivity), к нему будет добавлено имя пакета, указанное в элементе <manifest>

После публикации приложения, вы не должны менять значение этого атрибута (если только не установили атрибуте android:exported="false").

Атрибут не имеет значения по умолчанию и имя должно быть явно указано.

android:noHistory

Указывает должно ли явление удалять из стека и завершаться (вызывать метод `finish()`), когда пользователь покидает его и оно больше не видимо на экране.

`true` если должно завершаться, `false` если нет.

По умолчанию `false`.

Значение `true` означает, что явление “не оставляет след в истории”.

То есть явление не остается в стеке и пользователь не сможет к нему вернуться.

Если вы запускаете из такого явления другое явление для получения результата, метод `onActivityResult()` никогда не будет вызван!

Атрибут добавлен в API 3.

android:parentActivityName

Имя класса логического родителя явления.

Значение атрибута должно соответствовать имени класса, указанному в атрибуте android:name соответствующего элемента <activity>.

Система использует данный атрибут, чтобы знать какое явление должно быть запущено, если пользователь нажмет кнопку “назад”, а также для синтеза стека явлений с помощью TaskStackBuilder.

Для поддержки API 4 – 16, вы можете также объявить родителя с помощью элемента <meta-data>, который предоставляет значение "android.support.PARENT_ACTIVITY". Например:

```
<activity
    android:name="com.example.app.ChildActivity"
    android:label="@string/title_child_activity"
    android:parentActivityName="com.example.myfirstapp.MainActivity" >
    <!-- Parent activity meta-data to support API level 4+ -->
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value="com.example.app.MainActivity" />
</activity>
```

Атрибут добавлен в API 16

android:permission

Название разрешения, которое должен иметь клиент для запуска явления или заставить его реагировать на намерения.

Если компонент,зывающий методы `startActivity()` или `startActivityForResult()` не имеет указанного разрешения, намерение не будет доставлено в явление.

Если атрибут не установлен, применяется разрешение элемента `<application>`.

Если атрибут не установлен в обоих элементах, явление не защищено разрешением

android:process

Имя процесса, в котором явление должно быть запущено.

Как правило, все компоненты приложения запускаются в процессе по умолчанию, созданным для приложения и задавать этот атрибут не требуется.

Но если необходимо, вы можете задать другое имя процесса, что позволит распределить компоненты вашего приложения между несколькими процессами.

Если имя, указанное в атрибуте начинается с двоеточия “：“, новый процесс, частный для приложения, создается при необходимости и явления работают в нем.

Если имя процесса начинается с маленькой буквы, явление будет запущено в глобальном процессе с указанным именем, при условии, что оно имеет разрешение.

Это позволяет компонентам различных приложений использовать общий процесс, снижая расход ресурсов.

С помощью атрибута process элемента <application> можно задать имя процесса по умолчанию для всех компонентов приложения

android:relinquishTaskIdentity

Указывает должно ли явление освобождать идентификатор задачи для вышестоящих явлений в стеке.

Задача, в которой корневое явление имеет данный атрибут со значением true, заменяет базовое намерение на намерение следующего явления в задаче.

Если атрибут следующего явления также равен true, процесс будет продолжаться до тех пор, пока не встретится явление, атрибут которого равен false.

Значение по умолчанию false.

Установка атрибута в значение true также разрешает использовать ActivityManager.TaskDescription для изменения заголовка, цвета и иконки в списке последних приложений.

android:screenOrientation

Ориентация дисплея устройства для явления.

Возможные значения приведены в таблице ниже.

unspecified	Значение по умолчанию. Система самостоятельно выбирает ориентацию, поэтому может отличать от устройства к устройству.
behind	Такая же ориентация, как у явления, находящимся под ним в стеке.
landscape	Альбомная(ширина больше высоты)
portrait	Портретная (высота больше ширины)

reverseLandscape	Альбомная, но в обратном направлении. Добавлено в API 9.
reversePortrait	Портретная, но в обратном направлении. Добавлено в API 9.
sensorLandscape	Альбомная, может быть нормальной или обращенной в зависимости от датчика устройства. Добавлено в API 9.
sensorPortrait	Портретная, может быть нормальной или обращенной в зависимости от датчика устройства. Добавлено в API 9.
userLandscape	Альбомная, может быть нормальной или обращенной в зависимости от датчика устройства и его пользовательских настроек. Если пользователь заблокировал ориентацию на основе сенсора, ведет себя как landscape, иначе как sensorLandscape. Добавлена в API 18.

userPortrait	<p>Портретная, может быть нормальной или обращенной в зависимости от датчика устройства и его пользовательских настроек. Если пользователь заблокировал ориентацию на основе сенсора, ведет себя как <code>portrait</code>, иначе как <code>sensorPortrait</code>. Добавлена в API 18.</p>
sensor	<p>Ориентация задается датчиком положения и зависит от того, как пользователь держит устройство. Меняется, если пользователь поворачивает устройство. Некоторые устройства не поддерживают все четыре возможные ориентации. Для разрешения четырех направлений, используйте <code>fullSensor</code>.</p>
fullSensor	<p>Ориентация задается датчиком положения для всех четырех направлений. Работает также как <code>sensor</code>, но позволяет использовать четыре положения, даже если устройства не поддерживают такой режим. Добавлено в API 9.</p>

nosensor	Ориентация задается без использования физического датчика положения. Датчик игнорируется, поэтому изображение не будет вращаться при повороте устройства. Кроме этой особенности, система выбирает ориентацию также как в случае unspecified.
user	Текущая ориентация, выбранная пользователем.
fullUser	Если пользователь заблокировал сенсор, ведет себя также как user, иначе ведет себя как fullSensor и разрешает все 4 положения экрана. Добавлено в API 18.
locked	Блокирует ориентацию в текущем положении, каким бы оно ни было. Добавлено в API 18.

Когда вы используете одно из альбомных или портретных значений, это считается жестким требованием к ориентации, в котором явление запускается.

Таким образом, значение, которое вы укажете, позволяет Google Play допускать установку только на устройства, которые такую ориентацию поддерживают.

Например, если вы укажете `landscape`, `reverseLandscape` или `sensorLandscape`, приложение будет доступно только для устройств с альбомной ориентацией.

Однако вы также должны явно указывать в элементе `<uses-feature>`, что приложение требует альбомную или портретную ориентацию.

Например: `<uses-feature android:name="android.hardware.screen.portrait"/>`.

Помните, что такую фильтрацию обеспечивает только сервис Google Play (и другие сервисы, которые его поддерживают), и сама платформа Android не контролирует, можно ли установить приложение, если устройство не поддерживает некоторые ориентации.

android:stateNotNeeded

Указывает может ли явление быть уничтожено и перезапущено без сохранения состояния. true – если может, false – если требуется ссылка на предыдущее состояние.

Значение по умолчанию false.

Как правило, перед отключением явление сохраняет ресурсы с помощью метода `onSaveInstanceState()`.

Метод сохраняет текущее состояние в объект типа `Bundle`, который при перезапуске явления передается в метод `onCreate()`.

Если значение атрибута равно true, метод `onSaveInstanceState()` может быть не вызван и в метод `onCreate()` будет передано значение null вместо объекта `Bundle` – также как при первоначальном запуске явления.

Установка атрибута в значение true, гарантирует, что явление может быть перезапущено без остаточного состояния.

android:taskAffinity

Задача, с которой явление имеет родство.

Явления с таким же родством концептуально относятся к той же задаче (к тому же приложению, с точки зрения пользователя).

Родство задачи определяется значением родства его корневого явления.

Родство определяет две вещи – задачу, в которую перемещается явление (смотрите атрибут `allowTaskReparenting`) и задачу, в которой будет находиться явление, если оно запущено с флагом `FLAG_ACTIVITY_NEW_TASK`.

По умолчанию, все явления приложения имеют одинаковое родство.

Вы можете установить этот атрибут для группы явлений, а также определить место явления в различных приложениях, использующих одну и ту же задачу. Чтобы указать, что явление вообще не имеет родства с задачами, установите пустую строку.

Если этот атрибут не установлен, явление наследует родство приложения (смотрите элемент `<application>`).

Имя родства по умолчанию для приложения равно имени пакета, установленного в элементе `<manifest>`.

android:theme

Ссылка на ресурс стиля, который описывает тему для всего явления.

Этот атрибут автоматически устанавливает контекст явления для использования темы (смотрите `setTheme()`), а также может быть причиной анимации перед запуском явления.

Если атрибут не установлен, явление наследует тему от приложения (смотрите элемент `<application>`).

Если тема также не установлена для приложения, будет использоваться тема по умолчанию.

android:uiOptions

Расширенные опции для пользовательского интерфейса явления.

Может принимать одно из значений, приведенных в таблице:

Значение	Описание
none	Нет расширенных опций. Это значение по умолчанию.
splitActionBarWhenNarrow	Добавляет панель внизу экрана для отображения пунктов панели инструментов, если горизонтальное пространство ограничено (например в портретной ориентации на смартфонах). Вместо небольшого числа элементов на верхней панели инструментов, панель делится на верхнюю секцию навигации и нижнюю секцию для кнопок действий. Это гарантирует, что будет достаточно места не только для кнопок, но и для навигации и названия явления. Пункты меню не делятся между двумя панелями, они всегда находятся вместе.

Атрибут добавлен в API 14

android:windowSoftInputMode

Указывает как окно явления взаимодействует с окном, содержащим программную клавиатуру. Значение атрибута влияет на две вещи:

- Состояние программной клавиатуры – должна ли она быть скрыта или видима, когда явление получает фокус пользователя.
- Регулировка окна явления – должно ли оно становиться меньше, чтобы оставить место для клавиатуры, или панорамировать содержимое, чтобы текущий фокус был виден, когда часть окна скрыта под клавиатурой.

Атрибут может иметь одно из перечисленных в таблице значений, или сочетание одного значения "state..." с одним значением "adjust...". Установка нескольких значений одной группы имеет непредсказуемый результат. Отдельные значения разделяются вертикальной чертой "|". Например:
`<activity android:windowSoftInputMode="stateVisible|adjustResize" . . . >`

Элементы и атрибуты файла
Манифест

Соглашения файла Манифеста
Уровень API.

Функции манифеста
Проекты в JAVA(особенности)
Лекция 12+

<application>

СИНТАКСИС:

```
<application android:allowTaskReparenting=["true" | "false"]  
            android:allowBackup=["true" | "false"]  
            android:backupAgent="string"  
            android:banner="drawable resource"  
            android:debuggable=["true" | "false"]  
            android:description="string resource"  
            android:enabled=["true" | "false"]  
            android:hasCode=["true" | "false"]
```

android:hardwareAccelerated=["true" | "false"]
 android:icon="drawable resource"
 android:isGame=["true" | "false"]
 android:killAfterRestore=["true" | "false"]
 android:largeHeap=["true" | "false"]
 android:label="string resource"
 android:logo="drawable resource"
 android:manageSpaceActivity="string"
 android:name="string"
 android:permission="string"

```
    android:persistent=["true" | "false"]
    android:process="string"
    android:restoreAnyVersion=["true" | "false"]
    android:requiredAccountType="string"
    android:restrictedAccountType="string"
    android:supportsRtl=["true" | "false"]
    android:taskAffinity="string"
    android:testOnly=["true" | "false"]
    android:theme="resource or theme"
    android:uiOptions=["none" | "splitActionBarWhenNarrow"]
    android:vmSafeMode=["true" | "false"] >
    ...
</application>
```

СОДЕРЖИТСЯ В:

<manifest>

МОЖЕТ СОДЕРЖАТЬ:

<activity>

<activity-alias>

<meta-data>

<service>

<receiver>

<provider>

<uses-library>

ОПИСАНИЕ:

Описывает приложение. Элемент включает дочерние элементы, описывающие компоненты приложения и содержит атрибуты, влияющие на все компоненты.

Многие атрибуты (например, icons, label, permission, process, taskAffinity и allowTaskReparenting) устанавливают значения по умолчанию для соответствующих атрибутов дочерних элементов.

Другие (например debuggable, enabled, description и allowClearUserData) устанавливают значения для приложения в целом и не могут быть переопределены другими компонентами.

Атрибуты элемента <application>

android:allowTaskReparenting

Указывает могут ли явления приложения перемещаться из задачи, которая его запустила в родственную задачу, когда эта задача будет выведена на передний план: "true" если может перемещаться, "false" если должно оставаться в задаче, в которой было запущено.
Значение по умолчанию false

Атрибут может быть переопределен для каждого явления в элементах <activity>.

android:allowBackup

Указывает может ли приложение участвовать в резервном копировании и восстановлении. Если атрибут установлен в значение false, ни резервирование, ни восстановление не будут выполняться даже при полном резервировании системы с помощью adb. Значение по умолчанию true.

android:backupAgent

Название подкласса BackupAgent, который реализует агента резервирования. Атрибут должен содержать полное имя, включая название пакета (например, "com.example.project.MyBackupAgent").

Однако, если первым символом стоит точка ".", будет использовать название пакета, указанное в элементе <manifest>.

Атрибут не содержит значения по умолчанию.

android:banner

Ресурс типа `drawable` предоставляет расширенный графический баннер для связанного элемента.

Используется с тегом `<activity>` чтобы установить баннер по умолчанию для конкретного явления, или для тега `<application>`, чтобы установить баннер для всех явлений приложения.

Система использует баннер для представления приложения на домашнем экране Android TV.

Поскольку баннер отображается только на домашнем экране, он должен быть указан только для приложений, явления которых обрабатывают намерение с категорией `CATEGORY_LEANBACK_LAUNCHER`.

Значением атрибута должна быть ссылка на ресурс типа `drawable`, содержащий изображение (например "`@drawable/banner`"). Баннера по умолчанию нет.

android:debuggable

Указывает, может ли использовать отладка при запуске в пользовательском режиме. true если может. Значение по умолчанию false.

android:description

Описание приложения для пользователя. Атрибут должен содержать ссылку на строковые ресурс и не может содержать обычную строку. Не имеет значения по умолчанию.

android:enabled

Указывает, может ли система создавать компоненты приложения. true если может, false если нет.

Если установлено значение true, значение атрибута enabled каждого компонента указывает на его собственное значение.

Если указано false, создание всех компонентов запрещено и их собственное значение игнорируется.

Значение по умолчанию true.

android:hasCode

Указывает, содержит ли приложение какой-либо код, true если да. Если указано значение false, система не пытается загружать код приложения при запуске компонентов. Значение по умолчанию true.

Приложение не имеет собственного кода только в том случае, если оно использует исключительно встроенные компоненты, что случается исключительно редко.

android:hardwareAccelerated

Указывает должно ли использоваться аппаратное ускорение для отрисовки данного явления, true если должна, false если нет. Значение по умолчанию false.

Начиная с Android 3.0, аппаратное ускорение OpenGL доступно приложениям, для улучшения производительности для большинства распространенных 2D графических операций.

Если аппаратное ускорение разрешено, большинство операций с объектами Canvas, Paint, Xrefmode, ColorFilter, Shader и Camera ускоряются. В результате получается гладкая анимация, гладкое прокручивание и улучшенная отзывчивость во всем, даже в приложениях, которые явно не используют библиотеку OpenGL.

Поскольку для аппаратного ускорения требуется больше ресурсов, ваше приложение будет потреблять больше памяти.

Помните, что не все операции OpenGL 2D ускоряются. Если вы разрешили аппаратное ускорение, протестируйте ваше приложение, чтобы убедиться, что все отрисовывается без ошибок.

android:icon

Иконка приложения, является иконкой по умолчанию для всех его компонентов. Атрибут должен содержать ссылку на ресурс типа `drawable`. Не имеет значения по умолчанию.

android:isGame

Указывает, является ли приложение игрой. Система может группировать игры вместе и показывать отдельно от других приложений. Значение по умолчанию `false`.

android:killAfterRestore

Указывает, должно ли приложение завершаться после того, как его настройки были восстановлены при полном восстановлении системы. Восстановление одного пакета никогда не приводит к закрытию приложения.

Полное восстановление системы происходит однократно, как правило при первоначальном запуске телефона. Сторонние приложения, как правило, не нуждаются в данном атрибуте.

Значение по умолчанию `true`, это означает, что после обработки данных приложения во время восстановления, приложение будет закрыто.

android:largeHeap

Указывает, должен ли процесс приложения создавать большую кучу в Dalvik.
Применяется для всех процессов приложения.

Это относится только к первому приложению процесса; если вы используете общий идентификатор пользователя для того, чтобы разрешить приложениям использовать общий процесс, они всегда должны использовать эту опцию, иначе их выполнение имеет непредсказуемые результаты.

Для большинства приложений эта опция не нужна, и им необходимо сосредоточиться на снижении общего использования памяти, чтобы повысить производительность.

Включение опции также не гарантирует фиксированного увеличения доступной памяти, поскольку у некоторых устройств ограничен общий объем памяти.

Чтобы узнать доступный размер памяти во время выполнения, используйте методы `getMemoryClass()` или `getLargeMemoryClass()`.

android:label

Заголовок приложения, который виден пользователю и является заголовком по умолчанию для всех его компонентов. Часто используется вместе с иконкой.

Атрибут должен содержать ссылку на строковый ресурс, это поможет локализовать приложение. Однако во время разработки временно можно установить в качестве значения простую строку.

android:logo

Логотип по умолчанию, является логотипом по умолчанию для всех явлений приложения.

Атрибут должен содержать ссылку на ресурс типа `drawable`. Не имеет значения по умолчанию.

android:manageSpaceActivity

Полное имя подкласса `Activity`, который система может запускать, чтобы позволить пользователям управлять памятью, которую занимает приложение на устройстве. Это явление также должно быть объявлено в манифесте с помощью элемента `<activity>`.

android:name

Полное имя подкласса `Application`, который реализует приложение. При запуске приложения его экземпляр создается до создания других компонентов приложения.

Атрибут является необязательным и не требуется для большинства приложений. Если он не указан, система использует экземпляр базового класса.

Название разрешения, которое должен иметь клиент для взаимодействия с приложением. Атрибут удобен, чтобы установить общее разрешение для всех компонентов приложения

android:persistent

Указывает, должно ли приложение быть постоянно запущено, true если да. Значение по умолчанию false. Обычное приложение не должно использовать данный флаг; этот режим предназначен только для важных системных приложений.

android:process

Имя процесса, в котором запускаются все компоненты приложения. Каждый компонент может указать собственное имя процесса с помощью его атрибута process.

По умолчанию, процесс создается при запуске первого из компонентов приложения. Затем все компоненты запускаются в этом процессе. Имя процесса по умолчанию совпадает с именем пакета, указанным в элементе <manifest>.

Указав одинаковое значение данного атрибута для разных приложений, вы можете позволить им работать в одном процессе, но лишь в том случае, если они имеют одинаковый идентификатор пользователя и подписаны одним сертификатом.

Если имя, указанное в атрибуте начинается с двоеточия “：“, новый процесс, частный для приложения, создается при необходимости.

Если имя процесса начинается с маленькой буквы, приложение будет запущено в глобальном процессе с указанным именем.

Это позволяет компонентам различных приложений использовать общий процесс, снижая расход ресурсов.

android:restoreAnyVersion

Указывает, что приложение готово к восстановлению из любой резервной копии, даже если в ней была сохранена более новая версия приложения, чем установленная на устройстве в настоящее время. Установка атрибута в значение true позволит менеджеру попытаться восстановить данные, даже если они несовместимы. Используйте с осторожностью!

Значение по умолчанию false.

android:requiredAccountType

Указывает тип учетной записи, которая требуется приложению для выполнения функций. Если ваше приложение требует Account, значение атрибута должно соответствовать типу аутентификатора приложения (который указан с помощью AuthenticatorDescription), например "com.google".

Значение по умолчанию null указывает, что приложение может работать без каких-либо учетных записей.

Поскольку ограниченные профили в настоящее время не могут добавлять учетные записи, установка данного атрибута сделает ваше приложение недоступным для ограниченных профилей, если вы также не объявили атрибут `android:restrictedAccountType` с таким же значением.

Внимание: если данные учетной записи содержат личную информацию, очень важно, чтобы вы объявили этот атрибут, а также установили атрибут `android:restrictedAccountType` в значение `null`, так, чтобы ограниченные профили не могли использовать приложение и получить доступ к личной информации, которая принадлежит пользователю.

Атрибут добавлен в API 18

`android:restrictedAccountType`

Задает тип учетной записи, необходимой для работы приложения и указывает, что ограниченные профили имеют доступ к таким учетным записям, принадлежащим владельцу.

Если приложение требует Account и ограниченные профили имеют доступ к учетным записям, значение этого атрибута должно соответствовать типу аутентификатора (который указан с помощью `AuthenticatorDescription`), например `com.google`.

Значение по умолчанию `null` указывает, что приложение может работать без каких-либо учетных записей.

Внимание: указание этого атрибута позволяет ограниченным профилям использовать приложение с учетными записями, принадлежащими пользователю, которые могут содержать личную информацию.

Если учетная запись содержит персональную информацию, вы не должны использовать данный атрибут и вместо этого должны объявлять атрибут `android:requiredAccountType`, чтобы сделать приложение недоступным для ограниченных профилей.

Атрибут добавлен в API 18.

android:supportsRtl

Указывает, готово ли приложение к поддержке RTL(справа-на-лево) макетов.

Если установлено значение true и версия targetSdkVersion равна 17 или выше, будут использоваться RTL макеты.

Если значение равно false или targetSdkVersion равна 16 и ниже, API RTL будет проигнорировано и ваше приложение будет вести себя одинаково, независимо от направления макета (макеты всегда будут слева-на-право).

Значение по умолчанию false. Атрибут добавлен в API 17.

android:taskAffinity

Название родства для всех явлений приложения, за исключением тех, что указали свой собственный атрибут taskAffinity. Подробности смотрите в описании элемента <activity>.

По умолчанию, все явления приложения имеют одинаковое родство.

Имя родства по умолчанию для приложения равно имени пакета, установленного в элементе <manifest>.

android:testOnly

Указывает, используется ли приложение только в целях тестирования, например для поиска дыр в безопасности. Этот вид приложения может быть установлен только через adb.

android:theme

Содержит ссылку на ресурс стиля, объявляющий тему по умолчанию для всех явлений приложения. Явления могут переопределять это значение с помощью собственного атрибута theme

android:uiOptions

Расширенные опции для пользовательского интерфейса явлений.
Может принимать одно из значений, приведенных в таблице:

Значение	Описание
none	Нет расширенных опций. Это значение по умолчанию.
splitActionBarWhenNarrow	<p>Добавляет панель внизу экрана для отображения пунктов панели инструментов, если горизонтальное пространство ограничено (например в портретной ориентации на смартфонах). Вместо небольшого числа элементов на верхней панели инструментов, панель делится на верхнюю секцию навигации и нижнюю секцию для кнопок действий. Это гарантирует, что будет достаточно места не только для кнопок, но и для навигации и названия явления. Пункты меню не делятся между двумя панелями, они всегда находятся вместе.</p>

android:vmSafeMode

Указывает, желает ли приложение работать в безопасном режиме виртуальной машины. Значение по умолчанию false, добавлено: уровень API 1

Атрибут
добавлен в API
14.

<manifest>

СИНТАКСИС:

```
<manifest xmlns:android="schemas.android.com/apk/res/android"  
    package="string"  
    android:sharedUserId="string"  
    android:sharedUserLabel="string resource"  
    android:versionCode="integer"  
    android:versionName="string"  
    android:installLocation=["auto" | "internalOnly" | "preferExternal"] >  
    ...  
</manifest>
```

Не содержится в других элементах

ДОЛЖЕН СОДЕРЖАТЬ: <application>

МОЖЕТ СОДЕРЖАТЬ: <compatible-screens>, <instrumentation>, <permission>, <permission-group>, <permission-tree>, <supports-gl-texture>, <supports-screens>, <uses-configuration>, <uses-feature>, <uses-permission>, <uses-sdk>

ОПИСАНИЕ:

Корневой элемент файла AndroidManifest.xml.

Должен включать элемент <application> и обязательные атрибуты xmlns:android и package.

Атрибуты элемента <manifest>

xmlns:android

Описывает пространство имен Android. Всегда должен быть равен schemas.android.com/apk/res/android

package

Полное имя пакета приложения в стиле языка Java.

Имя должно быть уникальным и может включать большие и маленькие латинские буквы, цифры и символ подчеркивания “_”. Однако, имя пакета может начинаться только с буквы.

Чтобы избежать конфликтов с другими разработчиками, используйте доменное имя владельца в качестве основы для имени пакета (в обратном порядке). Например, приложения Google могут начинаться с com.google. Никогда не используйте пространство имен com.example при публикации приложения.

Имя пакета служит уникальным идентификатором приложения.

Это также имя по умолчанию для процесса приложения (смотрите атрибут process элемента <application>) и родство задачи по умолчанию (смотрите атрибут taskAffinity элемента <activity>).

Внимание: никогда не меняйте имя после публикации приложения, иначе система будет воспринимать его как новое приложение и пользователи не смогут обновить его до новой версии!

android:sharedUserId

Идентификатор пользователя Linux (Linux User ID), который может быть общим с другими приложениями. По умолчанию, Android ассоциирует каждое приложение с его собственным идентификатором пользователя.

Однако, если этот указано одинаковое значение данного атрибута у нескольких приложений, они могут использовать общий идентификатор пользователя – но приложения должны быть подписаны одним сертификатом.

Приложения с одинаковыми идентификаторами пользователя имеют доступ к данным друг друга и, при желании, могут работать в одном процессе.

android:sharedUserLabel

Метка для идентификатора пользователя. Атрибут должен содержать ссылку на строковый ресурс и не может содержать обычную строку.

Атрибут добавлен в API 3. Он имеет смысл только если установлен атрибут sharedUserId.

android:versionCode

Номер версии. Используется только для определения того, какая версия приложения является более поздней. Этот номер версии не виден для пользователей.

Атрибут должен содержать целочисленное значение

android:versionName

Версия, которая видна пользователям. Атрибут может содержать строку или ссылку на строковый ресурс. Стока служит только для того, чтобы отображаться для пользователей.

android:installLocation

Место для установки приложения по умолчанию. Может содержать значения, перечисленные в таблице:

Значение	Описание
internalOnly	Приложение должно быть установлено только во внутреннее хранилище. Если установлено это значение, приложение никогда не сможет переместить во внешнее хранилище. Если внутреннее хранилище переполнено, система не сможет установить приложение. Это значение по умолчанию.
auto	Приложение может быть установлено во внутреннее или внешнее хранилище, по умолчанию во внутреннее. Если внутреннее хранилище переполнено, система установит приложение во внешнее. Пользователь может перемещать приложение между хранилищами после его установки.
preferExternal	Предпочтительно устанавливать во внешнее хранилище. Но это не гарантирует, что система выполнит требование. Приложение может быть установлено во внутреннее хранилище, если внешнее переполнено или недоступно. Пользователь может перемещать приложение между хранилищами после его установки.

По умолчанию приложение устанавливается во внутреннее хранилище и не может быть установлено во внешнее, если вы не укажете для данного атрибута значение auto или preferExternal.

Когда приложение устанавливается на внешнее хранилище:

- .apk файл сохраняется во внешнем хранилище, но данные (например база данных) все еще находится во внутреннем.
- Контейнер, в котором сохранен .apk файл зашифрован ключем, который позволяет приложению работать только на устройстве, на которое было установлено приложение. (Приложение не будет работать, если пользователь вставит SD карту в другое устройство). Хотя на одном устройстве можно использовать несколько SD карт.
- По запросу пользователя, приложение может быть перемещено во внутреннее хранилище.

Пользователи также могут перемещать приложение из внутреннего хранилища во внешнее.

Однако, система не позволяет пользователям перемещать приложение на внешнее хранилище, если данный атрибут установлен в значение internalOnly.

Добавлен в API 8.

<permission>

СИНТАКСИС:

```
<permission android:description="string resource"  
          android:icon="drawable resource"  
          android:label="string resource"  
          android:name="string"  
          android:permissionGroup="string"  
          android:protectionLevel=["normal" | "dangerous" |  
                                 "signature" | "signatureOrSystem"] />
```

СОДЕРЖИТСЯ В:

<manifest>

ОПИСАНИЕ:

Объявляет разрешения безопасности, которые могут использоваться для ограничения доступа к определенным компонентам или функциям тех или иных приложений.

Атрибуты элемента <permission>

android:description

Описание разрешения, видимое для пользователей. Атрибут должен содержать ссылку на строковый ресурс и не может содержать обычную строку.

android:icon

Иконка для разрешения. Должен содержать ссылку на ресурс типа drawable.

android:label

Название разрешения, видимое для пользователей.

Может содержать обычную строку или ссылку на строковый ресурс (рекомендуется).

android:name

Название разрешения. Используется в коде для ссылки на разрешение – например, в атрибуте permission элемента <uses-permission>.

Название должно быть уникальным, поэтому используйте имя пакета в качестве префикса, например: “com.example.project.PERMITTED_ACTION”.

android:permissionGroup

Назначает группу для разрешения.

Атрибут содержит имя группы, которая должна быть объявлена с помощью элемента `<permission-group>` в каком-либо приложении.

Если атрибут не установлен, разрешение не принадлежит к какой-либо группе.

android:protectionLevel

Характеризует потенциальную опасность разрешения и указывает на процедуру, которой система должна следовать, решая выдавать разрешение или нет.

Доступные значения представлены в таблице:

Значение	Описание
normal	Значение по умолчанию. Разрешение с минимальным риском, которое дает доступ к изолированным функциям уровня приложения. Система автоматически выдает разрешение данного типа при установке, не спрашивая пользователя (хотя пользователь видит это разрешение в списке при установке приложения).
dangerous	Разрешение с более высоким риском, которое дает право доступа к личным данным пользователя или к управлению устройством. Поскольку данный тип включает потенциальный риск, система не может автоматически его выдать. Такие разрешения требуют подтверждения пользователя или другого подхода, предотвращающего их автоматическое применение.

signature	Разрешение, которое выдается только если приложение подписано тем же сертификатом, что и приложение, требующее это разрешение. Если подпись совпадает, система не запрашивает у пользователя подтверждения.
signatureOrSystem	Разрешение, которое выдается только для приложений, которые находятся в образе системы Android или подписаны тем же сертификатом, что и приложение, требующее разрешение. Пожалуйста избегайте использования этой опции, значение signature подходит для большинства приложений и работает независимо от места установки приложения. Значение signatureOrSystem используется только в крайних случаях, когда несколько разработчиков приложений, встроенных в образ системы нуждаются в использовании общих функций, т.к как разрабатывают их совместно.

<provider>

СИНТАКСИС:

```
<provider android:authorities="list"
    android:enabled=["true" | "false"]
    android:exported=["true" | "false"]
    android:grantUriPermissions=["true" | "false"]
    android:icon="drawable resource"
    android:initOrder="integer"
    android:label="string resource"
    android:multiprocess=["true" | "false"]
    android:name="string"
    android:permission="string"
    android:process="string"
    android:readPermission="string"
    android:syncable=["true" | "false"]
    android:writePermission="string" >
    ...
</provider>
```

СОДЕРЖИТСЯ В:

<application>

МОЖЕТ СОДЕРЖАТЬ:

<meta-data>

<grant-uri-permission>

<path-permission>

ОПИСАНИЕ:

Объявляет поставщика содержимого. Поставщик содержимого является подклассом ContentProvider, который обеспечивает структурированный доступ к данным, управляемым приложением.

Все поставщики содержимого вашего приложения должны быть объявлены с помощью элемента <provider> в файле манифеста, иначе они будут недоступны для системы и не смогут быть использованы.

Вам нужно объявить только поставщиком содержимого, которые являются частью вашего приложения. Не требуется объявлять поставщиков из других приложений.

Система Android хранит ссылки на поставщиков содержимого в соответствии с authority, которая является частью URI.

Например, вы хотите получить доступ к поставщику содержимого, который хранит информацию о медицинских работниках.

Чтобы это сделать, вызовите метод ContentResolver.query(), который среди прочих аргументов принимает URI, идентифицирующий поставщика

content://com.example.project.healthcareprovider/nurses/rn

Схема content: определяет, что URI указывает на поставщика системы Android. Authority com.example.project.healthcareprovider указывает на сам поставщик содержимого.

Android содержит список известных поставщиков и их authority.

Подстрока nurses/rn это путь (path), который использует поставщик для описания набора данных.

Когда вы определяете поставщика содержимого с помощью элемента <provider>, обратите внимание, что аргумент android:name содержит только authority и не содержит схему и путь.

Атрибуты элемента <provider>

android:authorities

Список из одного или нескольких URI authority, которые указывают на данные, предоставляемые поставщиком содержимого. Названия authority разделяются точкой с запятой ';;'.

Для избежания конфликтов, используйте стиль именования пакетов Java (например, com.example.provider.cartoonprovider).

Как правило, это имя подкласса ContentProvider, который реализует поставщика содержимого.

Атрибут не имеет значения по умолчанию. Обязательно должно быть указано хотя бы одно значение authority.

android:enabled

Указывает может ли поставщик быть создан системой, true – если может и false – если нет. Значение по умолчанию true.

Элемент <application> имеет собственный атрибут enabled, который применяется для всех компонентов приложения.

Атрибуты обоих элементов <application> и <provider> должны быть равны true (это их значение по умолчанию).

Если в одном из элементов будет указано значение false, поставщик не сможет быть создан.

android:exported

Указывает, доступен ли поставщик для использования другими приложениями:

- true: доступен для других приложений. Любое приложение может использовать URI поставщика содержимого для доступа к данным, с учетом разрешений для поставщика.
- false: недоступен для других приложений. Установите это значение для ограничения доступа к поставщику содержимого. Получить доступ смогут только приложения, имеющие такой же идентификатор пользователя.

Значение по умолчанию true для приложений, у которых один из атрибутов android:minSdkVersion или android:targetSdkVersion равен 16 или ниже, иначе значение по умолчанию false.

Вы можете установить значение атрибута, равное false и в дополнение к этому ограничить доступ с помощью разрешений (смотрите элемент <permission>).

android:grantUriPermissions

Указывает может ли выдаваться временное разрешение тому, кто обычно не имеет разрешения на доступ к данным поставщика содержимого.

Если установлено true, разрешение может быть выдано для любых данных поставщика, если false – разрешение может быть выдано только для набора данных, указанных в элементах <grant-uri-permission>. Значение по умолчанию false.

Другими словами, это способ выдачи компонентам приложения одноразового разрешения на доступ к защищенным данным.

Например, если e-mail сообщение содержит вложение, почтовый клиент может вызвать соответствующее приложение для его просмотра, несмотря на то, что это приложение не имеет общего разрешения для просмотра всех данных поставщика содержимого.

В таких случаях разрешение выдается с помощью установки флагов FLAG_GRANT_READ_URI_PERMISSION и FLAG_GRANT_WRITE_URI_PERMISSION в намерении, которой запускает компонент. К примеру, почтовый клиент может установить флаг FLAG_GRANT_READ_URI_PERMISSION для намерения и передать его в метод Content.startActivity().

Разрешение будет установлено для URI, указанного в намерении.

Если вы включаете данную функцию, установив данный атрибут в true или с помощью элементов <grant-uri-permission>, вы должны вызывать метод Context.revokeUriPermission(), когда URI удаляется из поставщика

android:icon

Иконка для поставщика содержимого.

Атрибут должен содержать ссылку на ресурс типа `drawable`.

Если атрибут не установлен, будет использоваться иконка приложения, указанная в элементе `<application>`.

android:initOrder

Порядок, в котором поставщик содержимого должен создаваться, относительно других поставщиков того же процесса.

Атрибут используется при наличии зависимости между поставщиками содержимого и гарантирует для каждого из них, что они создаются в порядке, предусмотренном этими зависимостями.

Атрибут должен содержать целочисленное значение.

Чем выше номер, тем раньше запускается поставщик.

android:label

Заголовок поставщика, видимый для пользователя.

Если атрибут не установлен, используется заголовок приложения, установленный в элементе `<application>`.

Атрибут должен содержать строку или ссылку на строковый ресурс (рекомендуется).

android:multiprocess

Указывает может ли экземпляр поставщика ресурсов быть создан в другом процессе. true если экземпляры могут работать в разных процессах, false если нет. Значение по умолчанию false.

Обычно поставщики процессов создаются в процессе приложения, в котором они объявлены.

Однако, если этот атрибут установлен в значение true, система может создавать экземпляры в любом клиентском процессе, где требуется взаимодействие, избегая таким образом расходов на взаимодействие между процессами.

android:name

Имя класса (подкласс ContentProvider), который реализует поставщика содержимого. Это должно быть полное имя, включая название пакета (например com.example.project.TransportationProvider).

Однако, если первым символом стоит точка, будет использоваться имя пакета, указанное в элементе <manifest>.

Атрибут не имеет значения по умолчанию.

android:permission

Название разрешения, которое должны иметь клиенты для чтения или записи данных поставщика содержимого.

Атрибут предоставляет удобный способ указать разрешение для записи и чтения в одном месте. Однако, атрибуты `readPermission` и `writePermission` имеют над ним приоритет.

android:process

Имя процесса, в котором поставщик содержимого должен быть запущен.

Как правило, все компоненты приложения запускаются в процессе по умолчанию, созданным для приложения и задавать этот атрибут не требуется.

Но если необходимо, вы можете задать другое имя процесса, что позволит распределить компоненты вашего приложения между несколькими процессами.

Если имя, указанное в атрибуте начинается с двоеточия “：“, новый процесс, частный для приложения, создается при необходимости и поставщики работают в нем.

Если имя процесса начинается с маленькой буквы, поставщик будет запущен в глобальном процессе с указанным именем, при условии, что оно имеет разрешение.

Это позволяет компонентам различных приложений использовать общий процесс, снижая расход ресурсов.

android:readPermission

Разрешение, необходимое клиентам для чтения поставщика содержимого.

android:syncable

Указывает должны ли синхронизироваться данные между поставщиком содержимого и сервером. true если должны, false если нет.

android:writePermission

Разрешение, необходимое клиентам для внесения изменений в данные, которые контролирует поставщик содержимого.

<receiver>

СИНТАКСИС:

```
<receiver android:enabled=["true" | "false"]  
        android:exported=["true" | "false"]  
        android:icon="drawable resource"  
        android:label="string resource"  
        android:name="string"  
        android:permission="string"  
        android:process="string" >  
  
    ...  
    </receiver>
```

СОДЕРЖИТСЯ В:

```
<application>
```

МОЖЕТ СОДЕРЖАТЬ:

```
<intent-filter>  
<meta-data>
```

ОПИСАНИЕ:

Объявляет широковещательный приемник (подкласс `BroadcastReceiver`) как компонент приложения.

Широковещательный приемник позволяет приложению получать намерения, которые рассыпает система или другие приложения, даже когда остальные компоненты приложения не запущены.

Существует два способа познакомить систему с приемником: один из них – объявить его в манифесте с помощью данного элемента.

Второй – создать приемник в коде и зарегистрировать с помощью метода `Context.registerReceiver()`.

Смотрите подробности в описании класса `BroadcastReceiver`.

Атрибуты элемента <receiver>

android:enabled

Указывает может ли приемник быть создан системой, true – если может и false – если нет. Значение по умолчанию true.

Элемент <application> имеет собственный атрибут enabled, который применяется для всех компонентов приложения.

Атрибуты обоих элементов <application> и <receiver> должны быть равны true (это их значение по умолчанию). Если в одном из элементов будет указано значение false, приемник не сможет быть создан.

android:exported

Указывает, может ли приемник получать сообщения от других приложений: true если может.

Если указано значение false, приемник может получать сообщения только от приложений, имеющих такой же идентификатор пользователя.

Значение по умолчанию зависит от того, содержит ли приемник фильтры намерений.

Отсутствие фильтров подразумевает, что приемник предназначен только для внутреннего пользования (если, конечно, другие не знают его имя).

В таком случае значение по умолчанию "false". С другой стороны, наличие хотя бы одного фильтра означает, что приемник предназначен для внешнего применения, и значение по умолчанию равно true.

Этот атрибут не единственный способ ограничить действие других приложений на приемник.

Вы также должны использовать разрешения для ограничения использования приемника внешними объектами.

android:icon

Иконка приемника. Атрибут должен содержать ссылку на ресурс типа `drawable` с изображением.

Если иконка для приемника не установлена, будет использована иконка, указанная в атрибуте `android:icon` элемента `<application>`.

Иконка, указанная в этом атрибуте, является также иконкой по умолчанию для всех фильтров намерений (смотрите элемент `<intent-filter>`).

android:label

Заголовок приемника, который виден пользователю. Если атрибут не установлен, будет использоваться значение атрибута `label` элемента `<application>`.

Заголовок, указанный в атрибуте, является также заголовком по умолчанию для всех фильтров намерений (смотрите элемент `<intent-filter>`).

Атрибут должен содержать ссылку на строковый ресурс, это поможет локализовать приложение.

Однако во время разработки временно можно установить в качестве значения простую строку.

android:name

Имя класса (подкласс BroadcastReceiver), который реализует данный широковещательный приемник.

Значение атрибута должно содержать полное имя, включая название пакета (например, com.example.project.ReportReceiver).

Тем не менее, в качестве сокращения, если первый символ названия точка (например, .ExtracurricularActivity), к нему будет добавлено имя пакета, указанное в элементе <manifest>

После публикации приложения, вы не должны менять значение этого атрибута (если только не установили атрибуте android:exported="false").

Атрибут не имеет значения по умолчанию и имя должно быть явно указано.

android:permission

Название разрешения, которое должен иметь компонент для отправки сообщения приемнику.

Если атрибут не установлен, применяется разрешение элемента <application>.

Если атрибут не установлен в обоих элементах, приемник не защищен разрешением.

android:process

Имя процесса, в котором приемник должен быть запущен. Как правило, все компоненты приложения запускаются в процессе по умолчанию, созданным для приложения и задавать этот атрибут не требуется.

Но если необходимо, вы можете задать другое имя процесса, что позволит распределить компоненты вашего приложения между несколькими процессами.

Если имя, указанное в атрибуте начинается с двоеточия “：“, новый процесс, частный для приложения, создается при необходимости и приемник работает в нем.

Если имя процесса начинается с маленькой буквы, приемник будет запущен в глобальном процессе с указанным именем, при условии, что оно имеет разрешение.

Это позволяет компонентам различных приложений использовать общий процесс, снижая расход ресурсов.

С помощью атрибута process элемента <application> можно задать имя процесса по умолчанию для всех компонентов приложения.

<service>

СИНТАКСИС:

```
<service android:enabled=["true" | "false"]  
        android:exported=["true" | "false"]  
        android:icon="drawable resource"  
        android:isolatedProcess=["true" | "false"]  
        android:label="string resource"  
        android:name="string"  
        android:permission="string"  
        android:process="string" >  
  
    ...  
    </service>
```

СОДЕРЖИТСЯ В:

```
<application>
```

МОЖЕТ СОДЕРЖАТЬ:

```
<intent-filter>  
<meta-data>
```

ОПИСАНИЕ:

Объявляет сервис (подкласс Service) как компонент приложения.

В отличие от явлений, сервисы не предоставляют пользовательский интерфейс.

Они используются для длительных фоновых операций или взаимодействия с API, которые могут быть вызваны другими приложениями.

Все сервисы приложения должны быть объявлены в манифесте с помощью элемента <service>, иначе они будут невидимы для системы и никогда не будут запущены.

Атрибуты элемента <service>

android:enabled

Указывает может ли сервис быть создан системой, true – если может и false – если нет. Значение по умолчанию true.

Элемент <application> имеет собственный атрибут enabled, который применяется для всех компонентов приложения.

Атрибуты обоих элементов <application> и <service> должны быть равны true (это их значение по умолчанию). Если в одном из элементов будет указано значение false, сервис не сможет быть создан.

android:exported

Указывает, могут ли компоненты других приложений вызывать сервис и взаимодействовать с ним: true если может.

Если указано значение false, только компоненты приложения или приложения с таким же идентификатором пользователя смогут работать с сервисом.

Значение по умолчанию зависит от того, содержит ли сервис фильтры намерений. Отсутствие фильтров подразумевает, что сервис предназначен только для внутреннего пользования (если, конечно, другие не знают его имя).

В таком случае значение по умолчанию "false". С другой стороны, наличие хотя бы одного фильтра означает, что сервис предназначен для внешнего применения, и значение по умолчанию равно true.

Этот атрибут не единственный способ ограничить действие других приложений на сервис. Вы также должны использовать разрешения для ограничения использования сервиса внешними объектами.

android:icon

Иконка сервиса. Атрибут должен содержать ссылку на ресурс типа `drawable` с изображением.

Если иконка для сервиса не установлена, будет использована иконка, указанная в атрибуте `android:icon` элемента `<application>`.

Иконка, указанная в этом атрибуте, является также иконкой по умолчанию для всех фильтров намерений (смотрите элемент `<intent-filter>`).

android:isolatedProcess

Если значение равно `true`, сервис будет запущен в специальном процессе, изолированном от остальной системы и не имеет собственных разрешений. Взаимодействие возможно только через Service API (связывание и запуск).

android:label

Заголовок сервиса, который виден пользователю. Если атрибут не установлен, будет использоваться значение атрибута `label` элемента `<application>`.

Заголовок, указанный в атрибуте, является также заголовком по умолчанию для всех фильтров намерений (смотрите элемент `<intent-filter>`).

Атрибут должен содержать ссылку на строковый ресурс, это поможет локализовать приложение.

Однако во время разработки временно можно установить в качестве значения простую строку.

android:name

Имя класса (подкласс Service), который реализует данный сервис.

Значение атрибута должно содержать полное имя, включая название пакета (например, com.example.project.RoomService).

Тем не менее, в качестве сокращения, если первый символ названия точка (например, .ExtracurricularActivity), к нему будет добавлено имя пакета, указанное в элементе <manifest>

После публикации приложения, вы не должны менять значение этого атрибута (если только не установили атрибуте android:exported="false").

Атрибут не имеет значения по умолчанию и имя должно быть явно указано.

android:permission

Название разрешения, которое должен иметь компонент для запуска сервиса.

Если компонент, вызывающий методы startService(), bindService() или stopService() не получил разрешение, метод не будет работать и сервис не получит объект Intent.

Если атрибут не установлен, применяется разрешение элемента <application>.

Если атрибут не установлен в обоих элементах, сервис не защищен разрешением

android:process

Имя процесса, в котором сервис должен быть запущен.

Как правило, все компоненты приложения запускаются в процессе по умолчанию, созданным для приложения и задавать этот атрибут не требуется.

Но если необходимо, вы можете задать другое имя процесса, что позволит распределить компоненты вашего приложения между несколькими процессами.

Если имя, указанное в атрибуте начинается с двоеточия “：“, новый процесс, частный для приложения, создается при необходимости и сервис работает в нем.

Если имя процесса начинается с маленькой буквы, сервис будет запущен в глобальном процессе с указанным именем, при условии, что оно имеет разрешение.

Это позволяет компонентам различных приложений использовать общий процесс, снижая расход ресурсов.

`<uses-sdk>`

СИНТАКСИС:

```
uses-sdk android:minSdkVersion="integer"  
        android:targetSdkVersion="integer"  
        android:maxSdkVersion="integer" />
```

СОДЕРЖИТСЯ В:

`<manifest>`

ОПИСАНИЕ:

Позволяет указать совместимость приложения с одной или несколькими версиями платформы, с помощью уровней API.

Несмотря на свое название, этот элемент используется для указания номера уровня API, а не номера версии SDK или версии платформы Android.

Уровень API это одно целое число, он всегда связан с соответствующим номером платформы Android.

Атрибуты элемента <uses-sdk>

android:minSdkVersion

Целое число, указывающее на минимальный уровень API, который требуется приложению для работы.

Система предотвращает установку приложения, если уровень API на устройстве ниже, чем указанный в данном атрибуте. Необходимо всегда использовать данный атрибут.

Внимание: если вы не укажете данный атрибут, система будет использовать значение по умолчанию равное 1, которое означает, что приложение совместимо со всеми версиями Android.

Если приложение не поддерживает все версии (например использует API 3) и вы не указали значение minSdkVersion, при установке приложения на систему с API ниже 3, оно будет аварийно завершаться при попытке использовать недоступные функции вышестоящего API.

Поэтому всегда указывайте соответствующее значение уровня API для данного атрибута.

android:targetSdkVersion

Целое число, указывающее версию API, на которую нацелено приложение. Если атрибут не указан, используется значение, эквивалентное minSdkVersion.

Атрибуте сообщает системе, что вы протестировали приложение в указанной версии и система не должна разрешать работать приложению в режиме совместимости.

С выходом новых версий, меняется поведение и внешний вид системы.

Однако, если уровень API платформы выше версии, указанной в данном атрибуте, система разрешает использовать режим совместимости, чтобы приложение работало так, как вы ожидаете.

Вы можете отключить такое поведение, указав в качестве targetSdkVersion версию платформы, на которой работает приложение.

Например, установка значения 11 и больше позволит системе применить тему Хопо, при запуске на Android 3.0 и выше, а также отключает режим совместимости экрана при запуске на больших экранах (потому что API 11 неявно поддерживает большие экраны).

Есть много режимов совместимости, который система включает, основываясь на значении атрибута.

Для поддержки приложения каждой версией Android, необходимо увеличивать значение этого атрибута в соответствии с последним доступным уровнем API, а затем тщательно тестировать приложение на соответствующей платформе

Что такое уровень API?

Уровень API это целое число, которое идентифицирует уникальную ревизию API платформы Android.

Платформа Android предоставляет API, которое приложения могут использовать для взаимодействия с основной системой и включает в себя:

- набор пакетов и классов ядра.
- набор XML элементов и атрибутов для файла манифеста.
- набор XML элементов и атрибутов для объявления ресурсов.
- набор намерений.
- набор разрешений, которые требуют приложения, а также включены в системы.

Каждая последующая версия Android может включать обновление системных приложений.

Обновления фреймворка разрабатываются так, чтобы оставаться совместимыми с более ранними версиями.

То есть большинство изменений в API добавляют новые возможности.

При обновлении части API, старая часть помечается как устаревшая, но не удаляется, поэтому существующие приложения могут ее использовать.

Функции очень редко удаляются из API, и это связанно в основном с обеспечением безопасности. Все остальные части более ранних версий переносятся без изменений.

Фреймворк, который использует платформа Android задается с помощью целого числа, который называется уровень API.

Каждая версия Android поддерживает ровно один уровень API, хотя он включает также все более ранние версии (вплоть до API 1).

Первый выпуск Android соответствует API 1. Всю таблицу соответствия можно посмотреть в ресурсах интернета

Соглашения файла Манифеста

Некоторые общие соглашения и правила для всех элементов и атрибутов манифеста:

Элементы

Обязательными являются только элементы `<manifest>` и `<application>`, каждый из них должен присутствовать в файле, при этом только в одном экземпляре. Большинство других элементов может отсутствовать или присутствовать в нескольких экземплярах.

Все, что могут включать в себя элементы - это другие элементы.

Все значения устанавливаются только с помощью атрибутов, а не как символьные данные внутри элемента.

Элементы могут быть неупорядочены.

Например, `<activity>`, `<provider>` или `<service>` могут находиться в любой последовательности.

Иключение из правил - элемент `<activity-alias>` должен всегда идти после элемента `<activity>`, для которого он создает псевдоним.

Атрибуты

Формально все атрибуты являются необязательными.

Однако, некоторые из них должны быть указаны, чтобы элемент выполнял свое предназначение.

Смотрите документацию, поистине необязательные атрибуты содержат значение по умолчанию, которое используется, если атрибут не указан.

За исключением некоторых атрибутов корневого элемента `<manifest>`, все имена атрибутов начинаются с префикса `android:` - например, `android:alwaysRetainTaskState`.

Поскольку префикс является универсальным, в документации для краткости он может пропускаться, когда речь идет о имени атрибутов.

Объявление имени класса

Многие элементы оперируют Java объектами, включая и сам элемент приложения (`<application>`) и соответствующие компоненты - явления (`<activity>`), сервисы (`<service>`), широковещательные приемники (`<receiver>`) и поставщики содержимого (`<provider>`).

Если вы создали подкласс, как вы почти всегда будете делать для всех компонентов, его имя указывается в атрибуте `name`.

Имя должно включать также название пакета.

Однако, для краткости, можно указать в качестве первого символа точку ".", тогда будет подставлено имя пакета, указанное в атрибуте `package` элемента `<manifest>`.

При запуске компонента, система создает экземпляр указанного подкласса. Если подкласс не указан, будет создан экземпляр базового класса

Множественные значения

Если может быть задано несколько значений, элемент обычно повторяется, а не указывается несколько значений в одном элементе. Например, фильтр намерений может содержать несколько значений.

Значения ресурсов

Некоторые атрибуты содержат значения, которые отображаются пользователям - например, иконка или заголовок явления.

Такие значения должны быть локализованы и всегда храниться в ресурсах. Значение ресурсов указывается в следующем формате:

`@[package:]type:name`

где имя пакета package может быть опущено, если ресурс находится в пакете того же приложения, type - это тип ресурса (например string или drawable), а name - имя ресурса.

Значения из темы выражаются таким же образом, но используется значок вопроса '?' вместо '@':

`?[package:]type:name`

Строковые значения

В качестве экранирующих символов строковых значений используется двойной бэкслэш "\\". Например, "\\n" для новой строки или "\\uxxxx" для символа таблицы Unicode.

ФУНКЦИИ манифеста

Фильтры намерений

Компоненты приложений запускаются с помощью намерений - объекта класса Intent.

Намерение включает информацию о данных, которые следует обработать, категории компонента, которые будет выполнять задачу и о других соответствующих инструкциях.

Android находит соответствующий компонент, создает новый экземпляр класса компонента и, если необходимо, передает в него объект Intent.

Компоненты описывают свои возможности, указывая какие намерения они способны обработать, с помощью фильтров намерения.

Таким образом, система Android будет знать, какие намерения может обработать компонент еще до его запуска.

Фильтры указываются в файле с помощью элемента <intent-filter>.

Компонент может иметь любое количество фильтров, каждый из них может описывать различные возможности компонента.

Если намерение содержит явное имя компонента для запуска, фильтры не имеют значения. Но если используется неявное намерение, оно должно пройти через один из фильтров компонента.

Заголовок и иконка

Некоторые элементы имеют атрибуты `icon` и `label` для иконки и текста, которые будут показаны пользователю.

Некоторые также имеют атрибут `description` для более подробного описания. Например, элемент `<permission>` имеет все три указанных атрибута, так что, при отображении разрешения пользователю, все они будут отображены на экране.

В любом случае, иконка и заголовок элемента будут являться также значениями по умолчанию для всех дочерних элементов.

Так иконка и заголовок элемента `<application>` будут установлены по умолчанию для элемента `<activity>` и в свою очередь для всех его элементов `<intent-filter>`.

Иконка и заголовок установленная для фильтра намерения, используется также для компонента, если он выполняет функцию, указанную в фильтре.

Например, фильтр со значением `android.intent.action.MAIN` и `android.intent.category.LAUNCHER` указывает, что приложение должно быть отображено в списке приложений. Следовательно, иконка и заголовок данного фильтра будут отображаться в списке приложений.

Разрешения

Разрешение - это ограничение доступа к части кода или данным на устройстве. Ограничение накладывается для защиты критически важных данных или кода, который может быть поврежден.

Каждое разрешение имеет уникальный идентификатор. Часто идентификатор сам по себе указывает на действие, которое ограничивает разрешение. Например:

android.permission.CALL_EMERGENCY_NUMBERS

android.permission.READ_OWNER_DATA

android.permission.SET_WALLPAPER

android.permission.DEVICE_POWER

Функция может быть защищена не более чем одним разрешением.

Если приложению нужен доступ к защищенной функции, оно должно указать это требование с помощью элемента `<uses-permission>`.

При установке приложения на устройство, установщик определяет выдается ли разрешение, проверяя сертификаты приложения и в некоторых случаях спрашивая пользователя.

Если разрешение выдано, приложение сможет использовать защищенные функции. Если нет, попытки приложения использовать защищенные функции будут просто отклонены без уведомления пользователя.

Приложение может также защищать свои собственные компоненты (явления, сервисы и.т.д) с помощью разрешений.

Для этого могут использовать либо свои собственные разрешения, либо объявленные системой Android, либо объявленные другими приложениями.

Новое разрешение объявляется с помощью элемента `<permission>`.

Отметим, что в данном примере разрешение `DEBIT_ACCT` не только объявлено с помощью элемента `<permission>`, но и задается его требование с помощью элемента `<uses-permission>`.

Использование разрешения требуется для того, чтобы другие компоненты приложения могли запустить защищенное явление, хотя защита накладывается для этого же приложения.

Если, в данном примере, атрибут `permission` был бы установлен для разрешения, объявленного в другом месте (например `android.permission.CALL_EMERGENCY_NUMBERS`), тогда бы не требовалось использовать элемент `<permission>`.

Однако, все еще необходимо было бы использовать элементе `<uses-permission>`.

Элемент `<permission-tree>` объявляет пространство имен для группы разрешений, которые будут объявлены в коде.

Элемент `<permission-group>` указывает заголовок набора разрешений.

Это влияет только на группировку разрешений при отображении пользователю.

Элемент `<permission-group>` не указывает какие разрешения относятся к группе, он просто дает ей имя.

Разрешения помещаются в группу с помощью атрибута `permissionGroup` элемента `<permission>`.

Библиотеки

Каждое приложение подключает стандартную библиотеку Android, которая включает основные пакеты для создания приложения (базовые классы, вроде Activity, Service, View, Intent и.т.д.).

Однако, некоторые пакеты находятся в других библиотеках.

Если ваше приложение использует один из таких пакетов, вы должны явно присоединить их к приложению.

Для этого используется элемент <uses-library>. Названия библиотек вы можете найти в документации к пакетам.

Проекты в JAVA

1. Пакеты

В стандартных Java-программах очень много классов. Сколько? Тысячи, десятки тысяч. А если еще посчитать, что программа использует различные библиотеки, которые содержат классы, написанные другими программистами, то количество классов легко может исчисляться миллионами!

Для всех этих миллионов, да и просто тысяч классов невозможно придумать уникальные имена.

Нет, ну конечно можно придумать имена типа A123, Б345, но если мы говорим о выборе для каждого класса хорошего имени, которое облегчает понимание этого класса (как String для строк, например), то даже тысяча уникальных имен — это большой труд.

Поэтому в Java все классы сгруппированы по пакетам.

Классы и их пакеты в Java по своей сути очень напоминают файлы и папки на компьютере.

Например, если вам нужно хранить на компьютере 10 документов, вы скорее всего просто будете хранить их в одной папке. А если документов тысячи (например, хранилище всех документов компании)?

Если бы нужно было хранить тысячи документов, решением было бы разработать несколько уровней папок с хорошими говорящими названиями. И в папке самого последнего уровня хранить документы, которые относятся к этой конкретной папке. Хорошие говорящие названия для документов тоже не помешают.

Фактически в Java это все и было сделано для классов.

Файлы с классами хранятся в разных директориях (папках), и полное название папки класса со всеми подпапками называется пакетом класса.

Пример:

Путь к файлу	Имя пакета	Имя класса
\com\javarush\tasks\Solution.java	com.javarush.tasks	Solution
\com\io\FileInputStream.java	com.io	FileInputStream
\java\util\ArrayList.java	java.util	ArrayList

Имя пакета, в отличие от имени папки, пишется через точку.

Т.е. если папка была \com\javarush\tasks\, ей будет соответствовать пакет com.javarush.tasks

2. Папка src

В Java принято все классы одной программы хранить в одной папке (и ее подпапках).

Такую папку обычно называют src (сокращение от source).

Такая папка называется корнем проекта (source root), и все пути для пакетов считаются от нее. Примеры:

Папки

c:\projects\data\my\src\com\javarush\
tasks\

d:\files\git\data\project\src\com\javar

Имя пакета

com.javarush.tasks

com.javarush.tasks

Программисты в такой ситуации скажут что-то типа «у нас есть проект по имени my, который расположен в папке c:\projects\data» или «у нас есть проект по имени project, который расположен в папке d:\files\git\data»

Лучше всегда класть классы в пакеты, а не в корень папки src. Когда классов мало, это ещё не проблема, но когда классов много, очень легко запутаться. Поэтому всегда создавайте классы только в пакетах.

В Java принято давать классам и пакетам осмысленные имена.

Многие компании выпускают свои библиотеки (набор классов) и, чтобы не было путаницы, называют пакеты этих классов по имени компании/сайта/проекта:

Имя пакета

org.apache.common

org.apache.tomcat

org.apache.util

com.oracle.jdbc

java.io

javax.servlet

com.ibm.websphere

com.jboss

Имя компании/проекта

Проект «Apache»

Компания «Oracle»

Компания Oracle, проект Java

Компания «IBM», проект WebSphere

Проект «JBoss»

3. Содержимое файла

Согласно стандарту языка Java, внутри файла с кодом должна быть записана информация об имени класса и имя его пакета. Схема стандарта приведена ниже:

```
package имя-пакета;  
public class Имя-класса  
{  
}
```

Имя пакета должно совпадать с именем папки, а имя файла — с именем публичного класса.

Если у вас есть файл ...\\src\\com\\project\\Service.java, значит у него внутри должно быть записано:

```
package com.project;  
public class Service  
{  
}
```

Импорт классов

Имя класса + имя пакета формируют так называемое полное уникальное имя класса.

Примеры:

Полное уникальное имя

java.io.FileInputStream

Имя пакета

java.io

Имя класса

FileInputStream

java.lang.String

java.lang

String

java.util.ArrayList

java.util

ArrayList

org.apache.tomcat.Servlet

org.apache.tomcat

Servlet

Cat

отсутствует

Cat

Полное имя класса всегда уникально в рамках одного проекта.

Ну вы же не можете создать два файла с одним и тем же именем в одной и той же папке.

Полные имена классов обычно либо длинные, либо очень длинные. А ведь каждый раз писать в коде длинное имя, например `java.util.ArrayList`, очень неудобно.

Поэтому в Java добавили возможность «импортировать классы».

Вы можете использовать в своем коде короткое имя класса, но сначала вам нужно будет объяснить компилятору, какое именно «полное уникальное имя класса» соответствует короткому имени.

Вдруг у вас в проекте несколько классов с таким именем.

Или сначала был один, а потом еще 15 добавилось...

Чтобы использовать короткое имя класса в своем коде, вам нужно добавить вот такую конструкцию в свой код:

`import полное-уникальное-имя-класса;`

Добавлять такое объявление нужно в самом начале класса, сразу после объявления package.

Пример:

```
package com.javarush.tasks.task01;  
import java.util.Scanner;  
import com.test.helper.special.ArrayList;  
public class Solution  
{  
    public static void main(String[] args)  
    {  
        Scanner console = new Scanner(System.in);  
        ArrayList list = new ArrayList();  
    }  
}
```

Благодаря тому, что мы импортировали два класса `java.util.Scanner` и `com.test.helper.special.ArrayList`, мы можем использовать их короткие имена в нашем коде.

И компилятор будет знать, какие именно классы использовать.

А вот как бы выглядел этот же код, если бы мы не использовали `import`:

```
package com.javarush.tasks.task01;  
public class Solution  
{  
    public static void main(String[] args)  
    {  
        java.util.Scanner console = new java.util.Scanner(System.in);  
        com.test.helper.special.ArrayList list = new com.test.helper.special.ArrayList();  
    }  
}
```

Кстати, если у вас в проекте есть два класса с именем Scanner, импортировать их оба в один файл с кодом не получится: для второго постоянно придется использовать длинное имя.

Допустим, у вас в коллективе есть Серега, и никаких проблем с общением, не возникает — все знают кто это.

Но если бы их было трое, чтобы их различать, пришлось бы использовать полные уникальные имена.

Примечание 1

Кстати, если вам лень добавлять много импортов в ваш класс, вы можете воспользоваться его версией для ленивых: вместо имени класса поставить звездочку:

```
import имя-пакета.*;
```

Таким образом, вы сможете использовать в вашем коде короткие имена всех классов из данного пакета.

Примечание 2

Все классы из пакета `java.lang` импортируются автоматически, поэтому вам не нужно писать для них `import`.

Один такой класс вы точно знаете: это класс... `java.lang.String`.

Да, да, тот самый класс `String`, который используется для работы со строками.

Создание первого приложения

Установка среды программирования

Лекция 13 +, 14 +

Создание первого мобильного приложения

Эти лекции помогут вам создать первое приложение для Android. Вы научитесь создавать проекты и запускать ваше приложение в режиме отладки. Вы так же узнаете об основах разработки приложения для Android, включая создание простого пользовательского интерфейса и обработку ввода информации пользователем.

Установка среды разработки

Перед тем как начать изучение, вам необходимо установить инструменты для разработки:

- Скачать Android Studio. (<https://developer.android.com/studio>)
- Скачать последнюю версию инструментов SDK, используя SDK Manager.

Примечание: Хотя большинство материала предполагает использование Android Studio, в некоторых случаях будем приводиться примеры работы с инструментами SDK в командной строке.

Данное занятие содержит пошаговые инструкции по созданию небольшого приложения, которые позволят вам узнать об основных понятиях, встречающихся при разработке Android приложений. Очень важно пройти занятие шаг за шагом от начала и до конца.

Android Studio

Android Studio — интегрированная среда разработки (IDE) для работы с платформой Android. Эта среда разработки, основанная на программном обеспечении IntelliJ IDEA от компании JetBrains, официальное средство разработки Android приложений. Данная среда разработки доступна для Windows, OS X и Linux.

Android Studio это свободное, кроссплатформенное и с открытым исходным кодом графическое приложение реализованное на Java и разработано со смещением которое будет использоваться для разработки всех видов приложений для мобильной операционной системы Android.

Распространение проходит в рамках проекта Android Tools от Google

The Android Studio Application является частью проекта Android Tools от Google, которая предоставляет несколько полезных и мощных инструментов для разработки приложений Android на нескольких платформах.

Среди этих инструментов, можно отметить несколько плагинов Eclipse, Android OS Emulator, и Android SDK (Software Development Kit), AVD (Android Virtual Disk) Manager, Hierarchyviewer, ddms, а также другие полезные утилиты командной строки.

Android Studio имеет простой в использовании, и интуитивно понятный графический интерфейс, который позволяет пользователю создавать новый проект, импортировать существующий проект, открыть существующий проект, проверить проекты из системы управления версиями, а также для настройки различных параметров и включает в себя встроенную документацию и обучающие программы.

Основные возможности

- Расширенный редактор макетов: WYSIWYG, способность работать с UI компонентами при помощи Drag-and-Drop
- Сборка приложений, основанная на Gradle.
- Различные виды сборок и генерация нескольких .apk файлов
- Рефакторинг кода
- Статический анализатор кода, позволяющий находить проблемы производительности, несовместимости версий и другое.
- Встроенный ProGuard и утилита для подписывания приложений.
- Шаблоны основных макетов и компонентов Android.
- Поддержка разработки приложений для Android Wear и Android TV.
- Встроенная поддержка Google Cloud Platform, которая включает в себя интеграцию с сервисами Google Cloud Messaging и App Engine.
- Начиная Android Studio 2.1 поддерживается Android N Preview SDK
- Начиная с версии Android Studio 2.1 способна работать с компилятором Jack, а также получила улучшенную поддержку Java 8 и усовершенствованную функцию Instant Run.
- Начиная с версии Platform-tools 23.1.0 для Linux поддерживается только 64bit версия

Установить Android Studio в Ubuntu/Linux mint и другие производные

Есть два способа установки этой среды разработки в вашу систему.

Первый способ, заключается в скачивании с официального сайта скрипта установщика, давайте скачаем

[Скачать Android Studio](#)

Но для установки нам потребуются некоторые библиотеки, а также Oracle Java

Для установки Oracle Java в Ubuntu, открываем терминал и вводим следующие команды

```
sudo add-apt-repository -y ppa:webupd8team/java  
sudo apt update  
sudo apt install oracle-java8-installer
```

Устанавливаем дополнительные библиотеки

```
sudo apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386 lib32z1 libbz2-1.0:i386
```

Далее если вы скачали Android Studio в директорию «Загрузки», то
переходим в эту директорию, и нам нужно переместить нашу программу в
/usr/local/, сделаем это с помощью команды

```
sudo mv ~/Загрузки/android-studio /usr/local/
```

Переходим в директорию «bin», и запускаем скрипт установки

```
cd /usr/local/android-studio/bin  
./studio.sh
```

Второй способ установки

Второй способ заключается в установке из репозитория, но также нам понадобится установить Java, если вы еще не установили.

Для установки Java в Ubuntu, открываем терминал и вводим следующие команды

```
sudo add-apt-repository -y ppa:webupd8team/java
```

```
sudo apt update
```

```
sudo apt install oracle-java8-installer
```

Далее добавим репозиторий для установки Android Studio

```
sudo add-apt-repository ppa:paolorotolo/android-studio
```

Обновляем список пакетов и начинаем установку

```
sudo apt update
```

```
sudo apt-get install android-studio
```

Установка на Ubuntu с помощью Ubuntu Make

Чтобы установить Android Studio на Ubuntu, можно воспользоваться Ubuntu Developer Tools Center, также известным как Ubuntu Make.

Чтобы установить Ubuntu Make на Ubuntu 16.04 и старше, нужно ввести в терминал следующую команду.

sudo apt install ubuntu-make

На всякий случай:

Если версия Ubuntu младше 16.04, то для установки Ubuntu Make нужно добавить репозиторий из персонального архива пакетов (PPA), введя следующие команды.

sudo add-apt-repository ppa:ubuntu-desktop/ubuntu-make

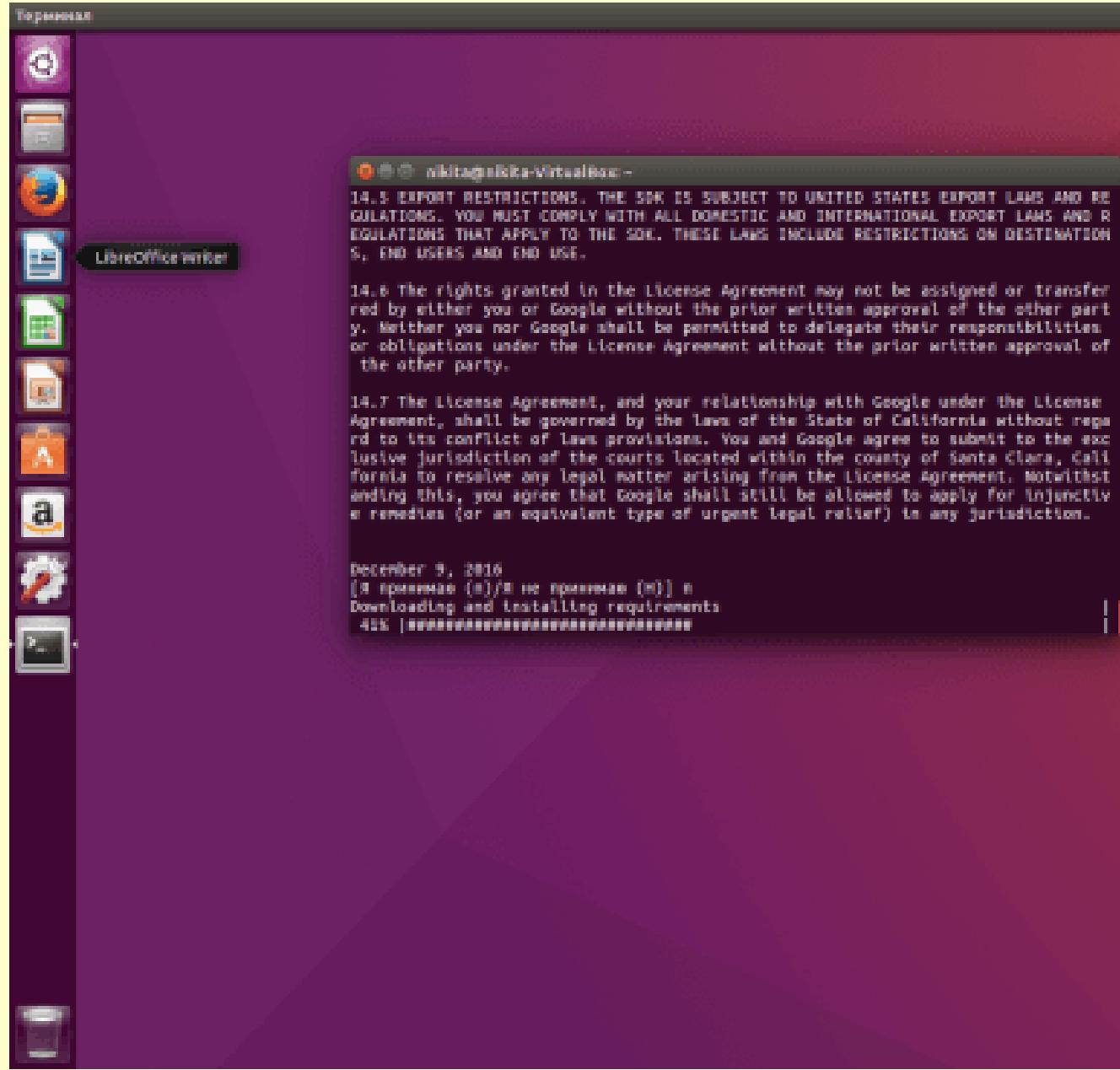
sudo apt-get update

После того, как Ubuntu Make установлен, можно приступить к установке Android Studio. Для этого нужно ввести следующую команду.

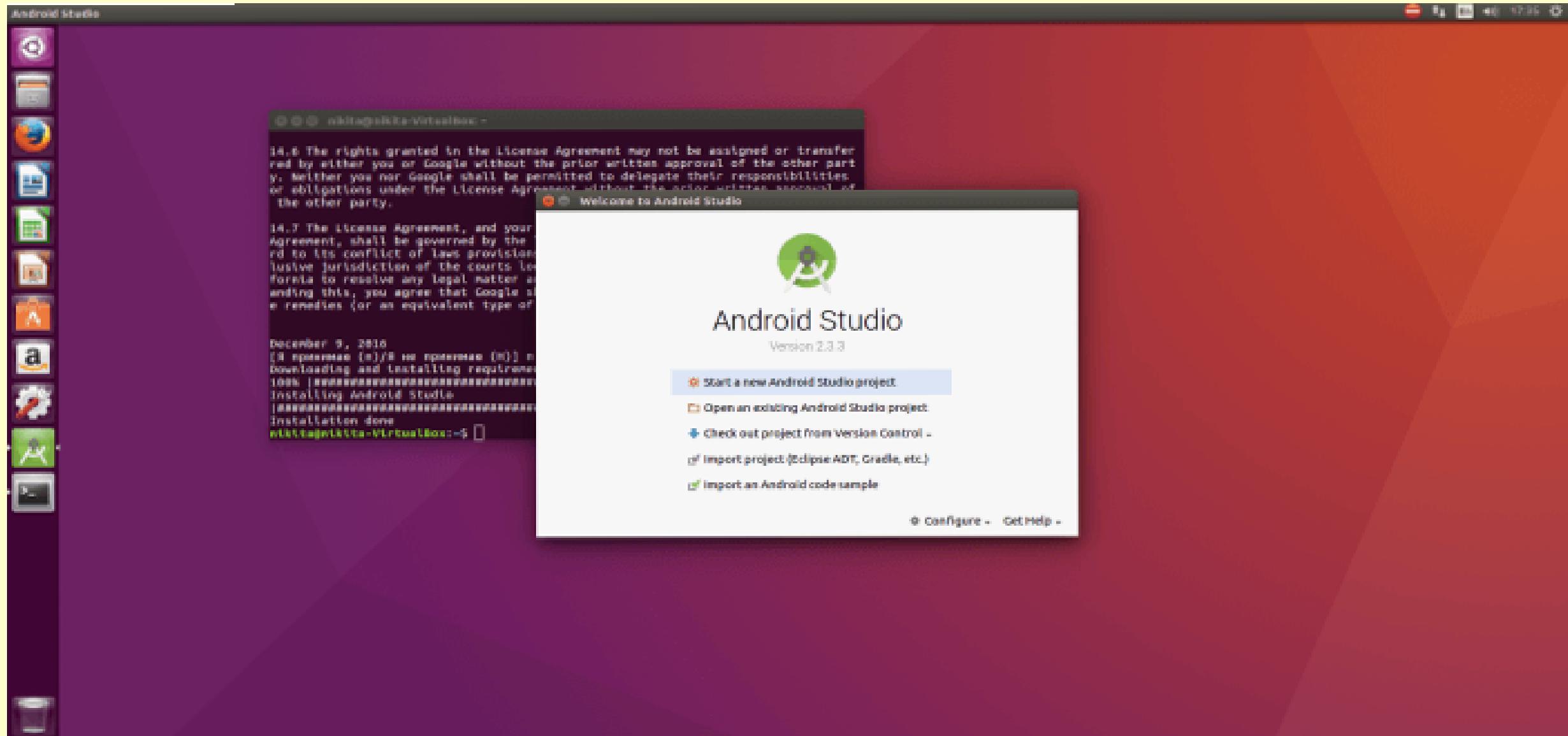
umake android

Система предложит выбрать путь, куда будет установлена Android Studio, принять лицензионное соглашение, и затем начнёт установку.

В каждом из перечисленных случаев вам потребуется заблаговременно установить виртуальную машину Java и использовать терминал для написания команд.



После того, как будет установлен SDK, можно начинать пользоваться Android Studio.

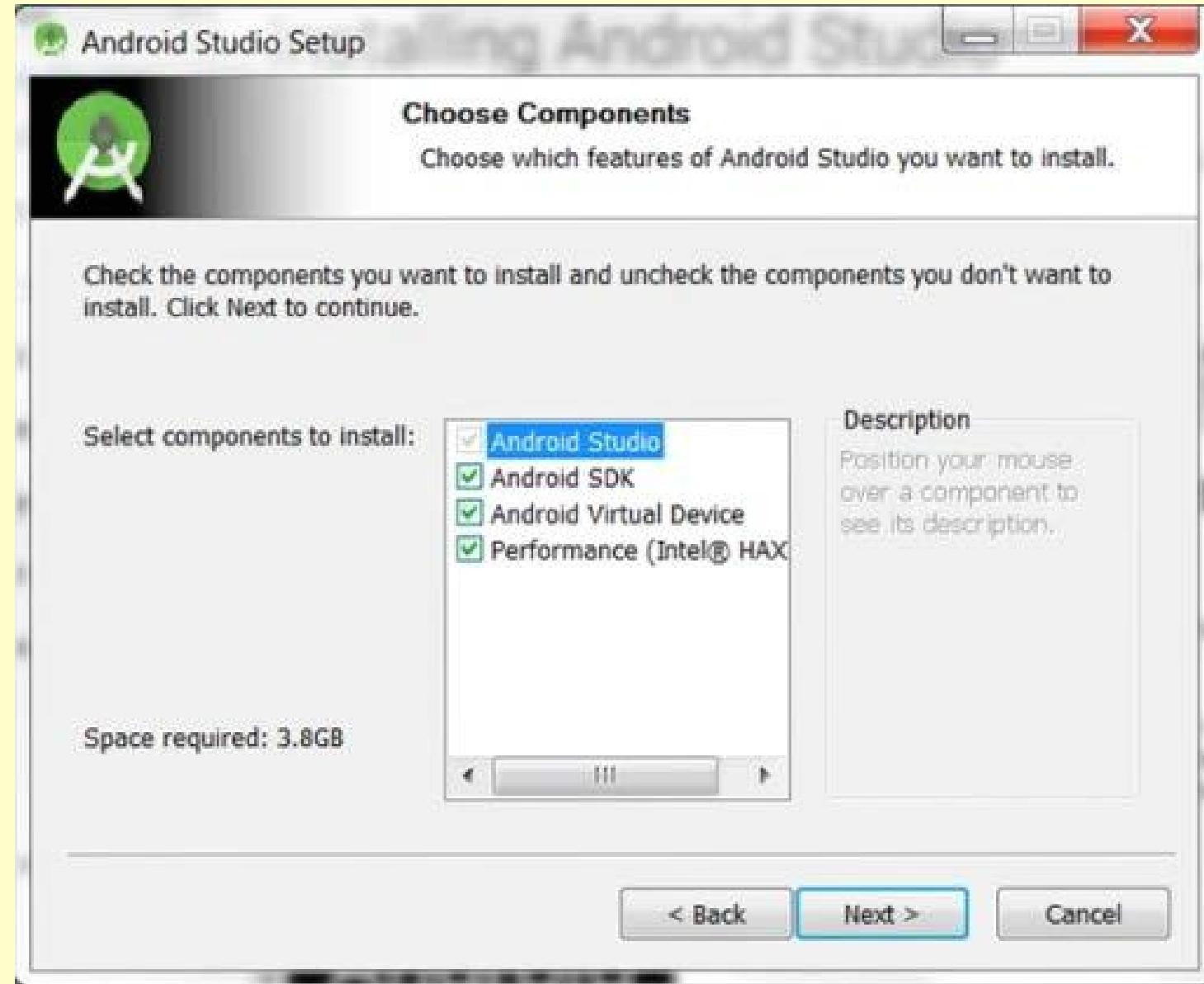


Инсталляция Android Studio под Windows

Перейдите по этой [ссылке](#) на сайт Андроид-разработчиков, чтобы инсталлировать Android Studio. Эта страница определит Вашу операционную систему автоматически.

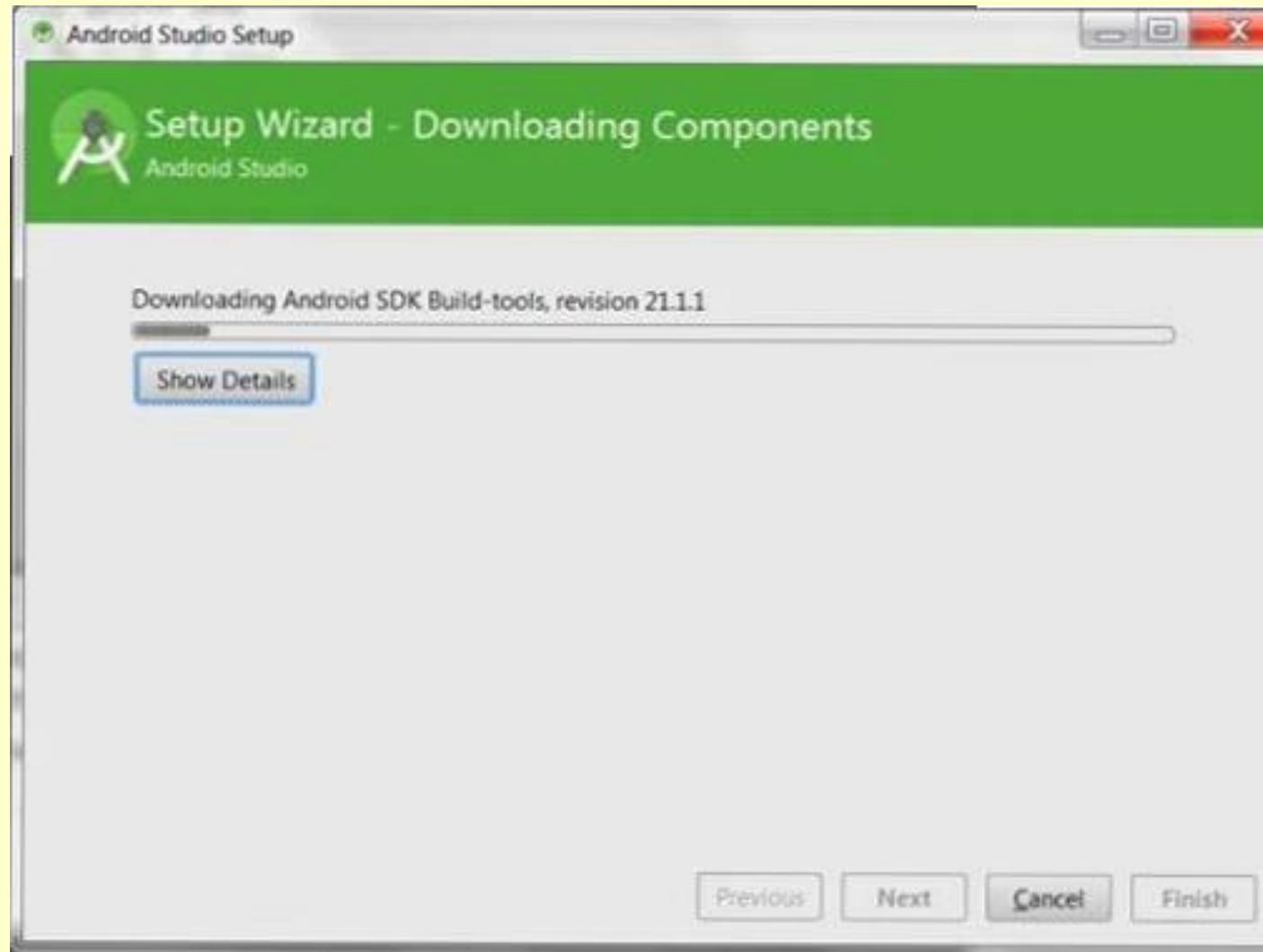


Когда доберетесь до этого экрана, убедитесь, что все компоненты выбраны.



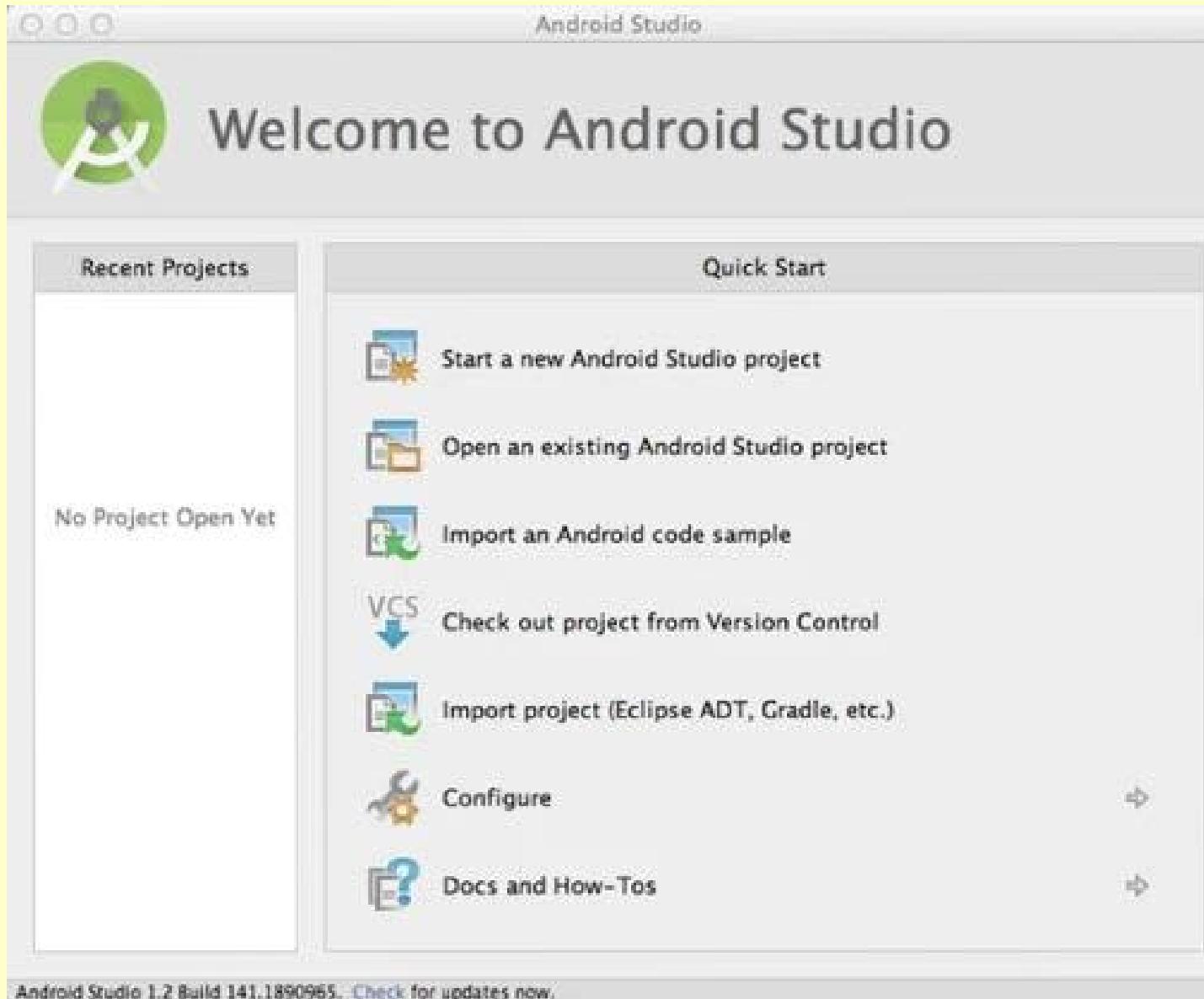
Мастер установки

После инсталляции Мастер Установки скачает все дополнительные компоненты. Наберитесь терпения, это займет некоторое время в зависимости от скорости интернет-соединения.



Установка завершена!

После завершения установки Вы увидите такое окно:



Установка среды разработки IntelliJ IDEA и Java

Используем Java 11 LTS версию. Это версия с длительной поддержкой, поэтому имеет смысл обновиться. Загрузите себе эту версию.

<https://adoptopenjdk.net>

Выбирайте “Other platforms”, чтобы загрузить архив

Версия Open JDK 11, JVM: HotSpot

Выберите свою операционную систему, например, windows и архитектуру, скорее всего у вас x64.

В списке снизу выберите вариант JDK (Java вместе с инструментами разработки) и zip архив.

[Build archive](#)[Nightly builds](#)

1. Choose a Version

- OpenJDK 8 (LTS)
- OpenJDK 9
- OpenJDK 10
- OpenJDK 11 (LTS)
- OpenJDK 12
- OpenJDK 13
- OpenJDK 14 (Latest)

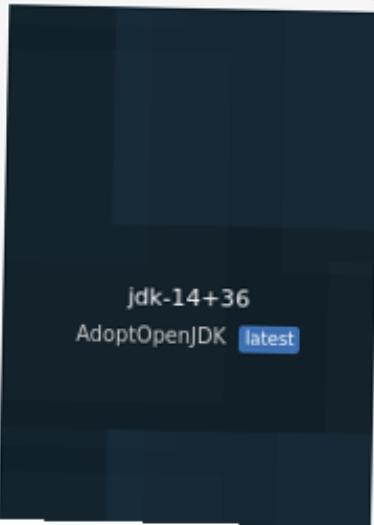
2. Choose a JVM

- HotSpot
- OpenJ9

[All Release Notes](#)

Operating System:

Architecture:

Windows
2008r2 or later

x64

Normal

[Checksum
\(SHA256\)](#) [!\[\]\(3831df74cdddf9eec94867bb3f5b8bb1_img.jpg\) .msi](#)
JDK - 189 MB

[Checksum
\(SHA256\)](#) [!\[\]\(451b841b61f83c5b19bab3b7c9470403_img.jpg\) .zip](#)
JDK - 212 MB

[Checksum
\(SHA256\)](#) [!\[\]\(994e93e3a581758c6af0833e79cec309_img.jpg\) .msi](#)
JRE - 40 MB

Раскройте архив в какое-то место, которое вы будете знать. У себя на линукс я сделал папку opt в домашней директрии. /home/ilya/opt/jdk-14+36. Вы можете раскрыть в C:\Program Files\jdk-14+36.

Мы умеем пользоваться Java без дополнительных инструментов, нам было достаточно только этого скаченного архива. Сейчас мы установим среду разработки, которая значительно упростит работу и даст много полезных возможностей.

IntelliJ IDEA

IntelliJ IDEA от фирмы JetBrains — интеллектуальная среда разработки, она понимает код, который вы пишете, подсказывает, что нужно написать дальше и дает содержательные советы по тому коду, который уже написан.

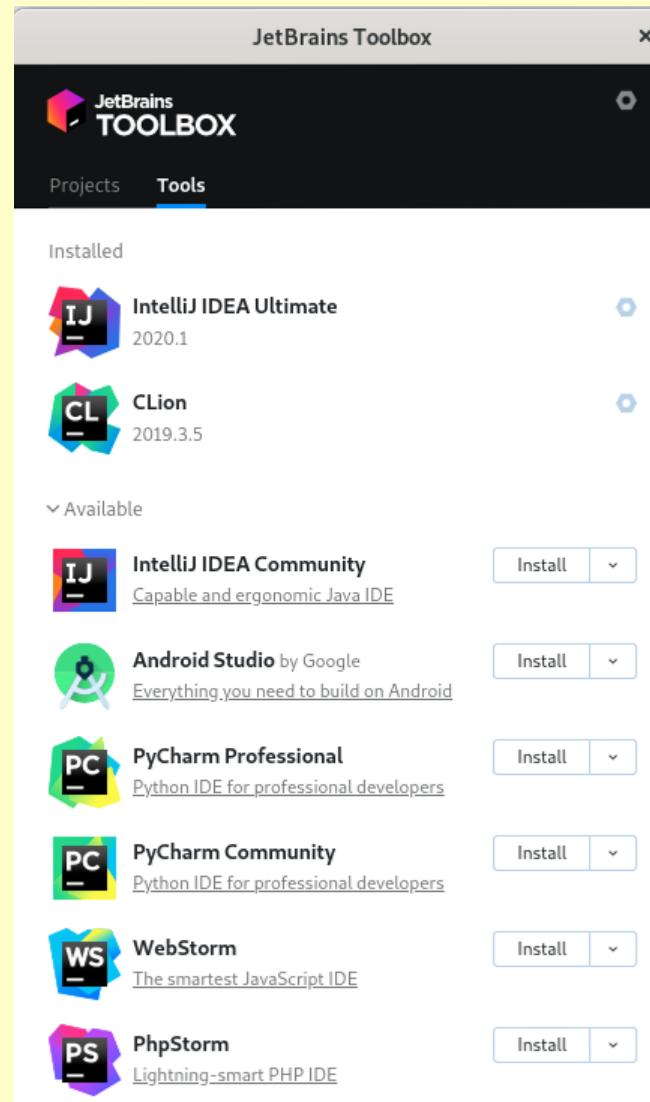
Профессиональные разработчики обязательно пользуются средой разработки.

Можно использовать для работы IntelliJ IDEA, а так же другие среды разработки (Eclipse, NetBeans из самых распространенных) и другие текстовые редакторы (Atom, Sublime и т.п.)

Среда разработки IntelliJ IDEA исторически была разработана для программирования на Java, но сейчас с ее помощью можно программировать практически на всех сколько-нибудь распространенных языках программирования. Вы можете установить только одну IntelliJ IDEA, и использовать ее и для Java, и для Python, и для HTML+CSS+Javascript, и для PHP, и для других языков. Есть отдельные сокращенные версии программы, например, PyCharm, которые подходят только для работы с Python. Они нужны, чтобы, во-первых, предложить более простой интерфейс, для тех, кому не нужно ничего из Java, во-вторых, они стоят дешевле.

Про стоимость. IntelliJ IDEA Community Edition (дословно, версия для сообщества) для Java и Python, или PyCharm можно использовать бесплатно. Т.е. на Java и Python с помощью инструментов JetBrains вы можете программировать бесплатно. Остальные программы, включая IntelliJ IDEA Ultimate Edition (полная версия) требует платной лицензии, но для студентов университетов и преподавателей эта лицензия доступна бесплатно. Достаточно корпоративного email адреса университета. Я рекомендую ее получить, потому что в IntelliJ IDEA Ultimate Edition есть возможность веб разработки (HTML, CSS, JavaScript), которую вы изучаете на других курсах, и другие возможности, которые могут пригодиться.

IDEA можно установить напрямую, но я прошу так не делать. Это усложнит ее обновление, вам придется периодически загружать новую версию вручную. Лучше воспользоваться программой JetBrains Toolbox, загрузите ее, установите, запустите, вы увидите что-то наподобие:



Справа сверху найдите шестеренку с настройками всей программы и уберите внутри “Run at login”, чтобы программа не запускалась сама при старте. Закройте настройки

Найдите в списке “IntelliJ IDEA Community” и установите ее кнопкой Install.

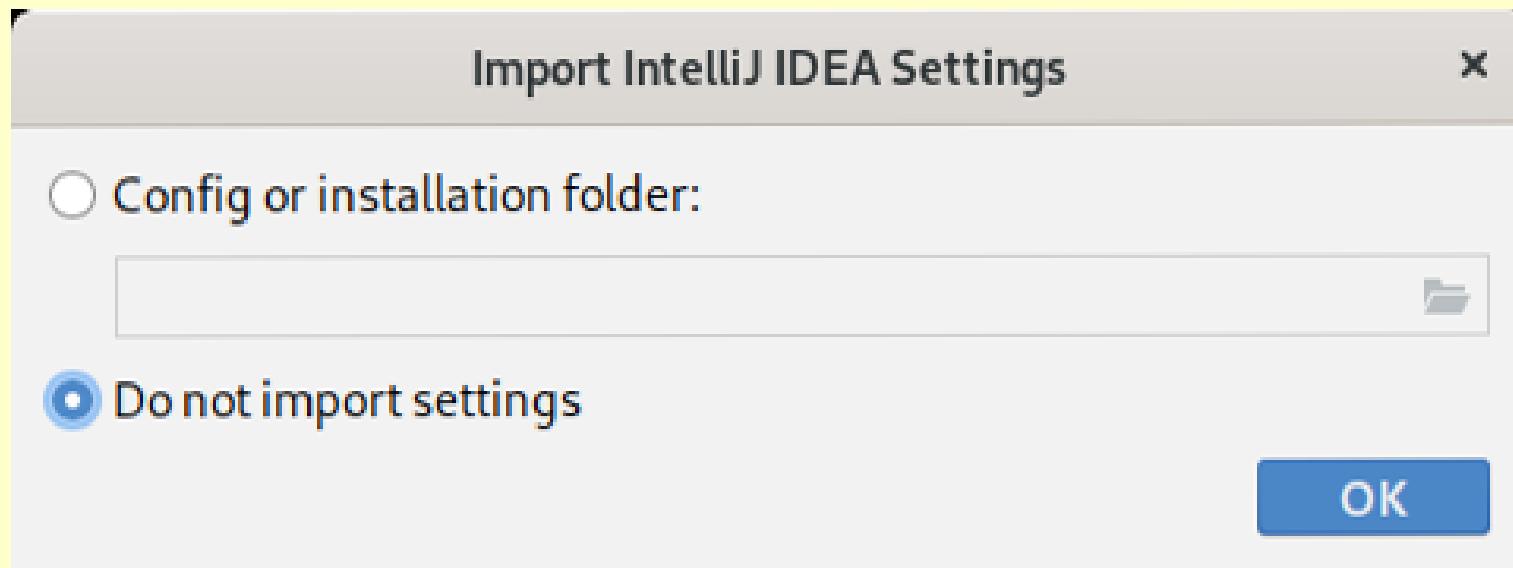
Вы можете установить Ultimate версию вместо Community. В Ultimate больше возможностей, некоторые из них полезны, но эта версия имеет больший размер и в ней больше пунктов меню, поэтому она может пугать своим перегруженным внешним видом. Кроме того, для версии Ultimate вам придется получить на сайте jetbrains студенческую лицензию, чтобы пользоваться IDEA бесплатно.

При необходимости обновить среду разработки в будущем, открывайте toolbox и нажимайте “upgrade”.

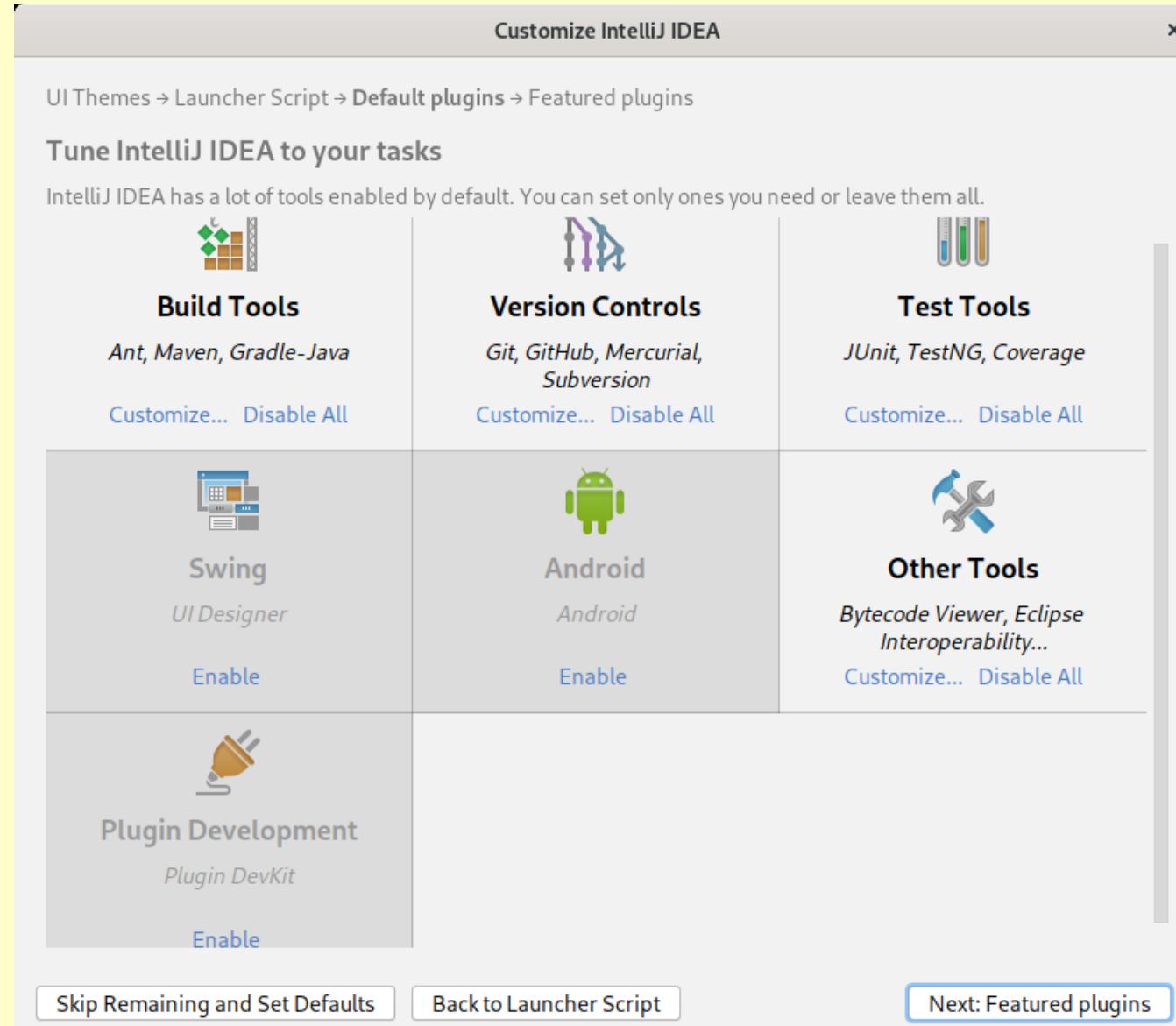
Первый запуск IDEA

Запускайте IDEA через программу Toolbox или, как обычно, из списка установленных программ.

При первом запуске вы увидите окно которое спрашивает, настраивать IDEA с нуля, или можно взять какие-то старые настройки. Скорее всего, старых настроек нет, поэтому выбирайте “do not import settings”.



Далее, выбирайте темную или светлую тему оформления, пропускайте экраны, пока не увидите окно выбора плагинов:

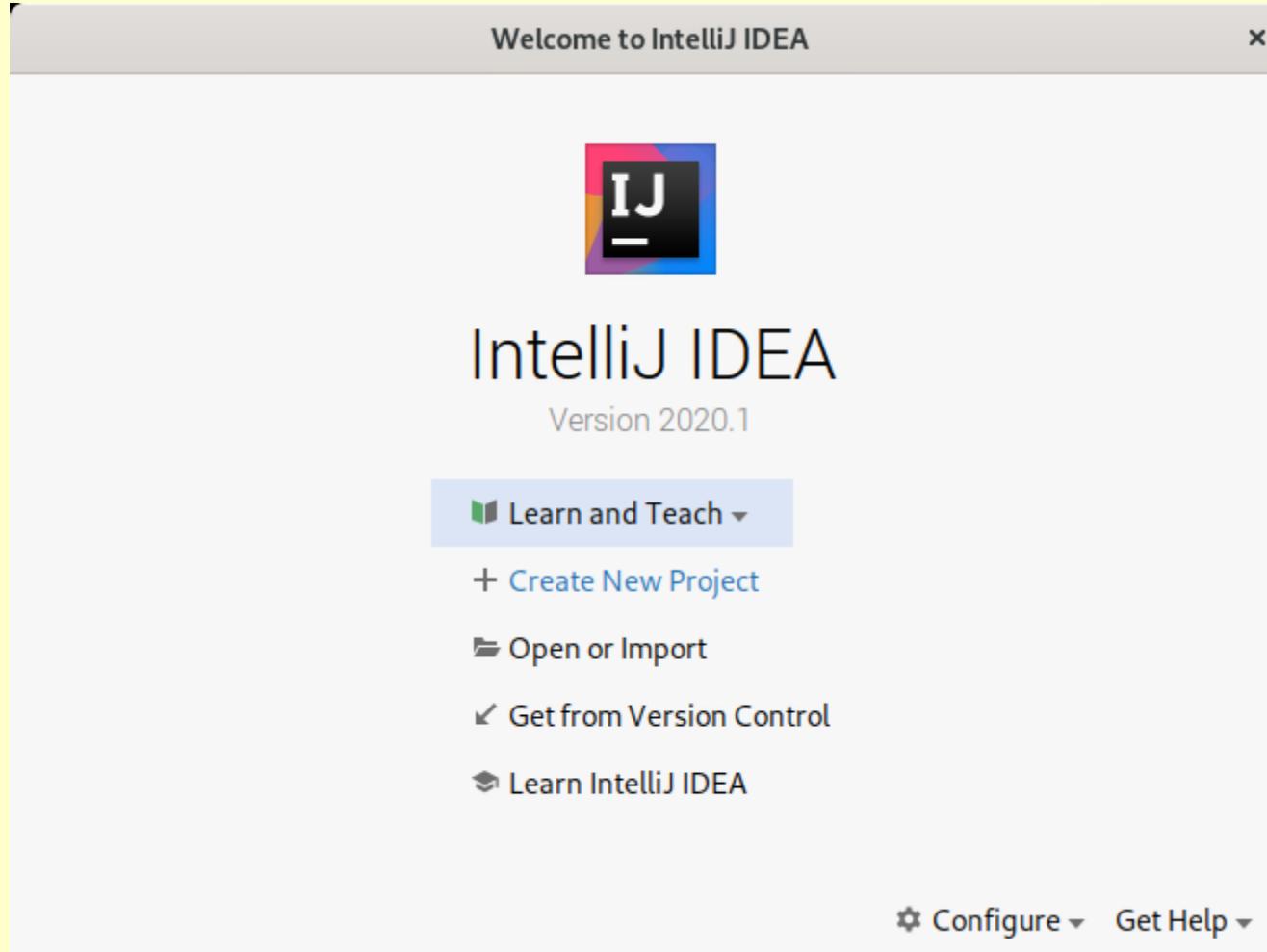


В нем отключите плагины, как я показал на картинке. Плагины лучше отключать, потому что, чем их больше, тем дольше запускается IDEA, и тем больше разных отвлекающих пунктов в меню.

На следующем окне тоже выбор плагинов, из них, возможно, вас заинтересуют EduTools и IDE Feature Trainer. Первый позволяет интегрироваться со Stepik для решения задач, второй я настоятельно рекомендую для изучения возможностей среды.

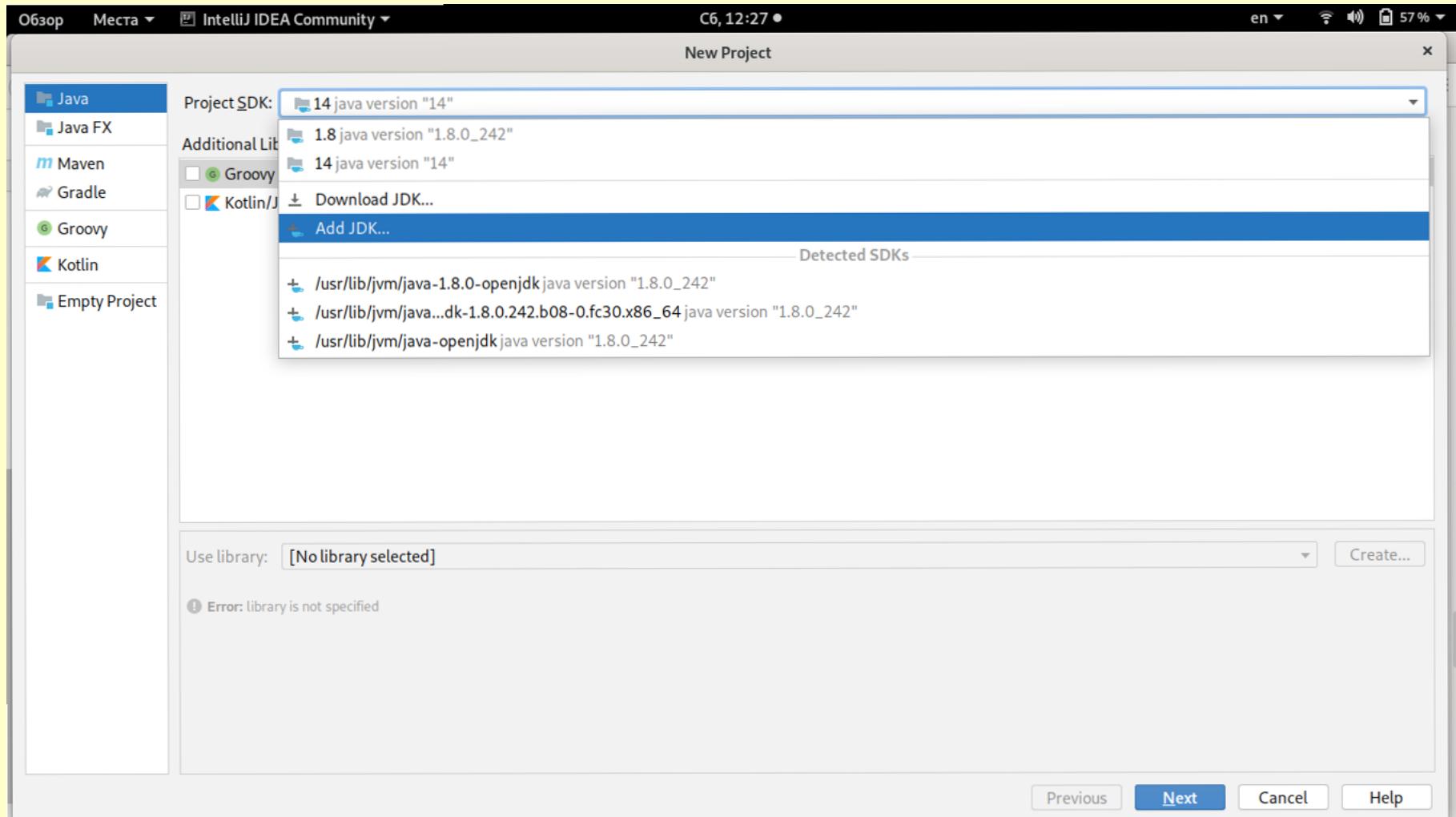
Создание проекта

Настройка закончена, после запуска IDEA вы увидите:
screenshot

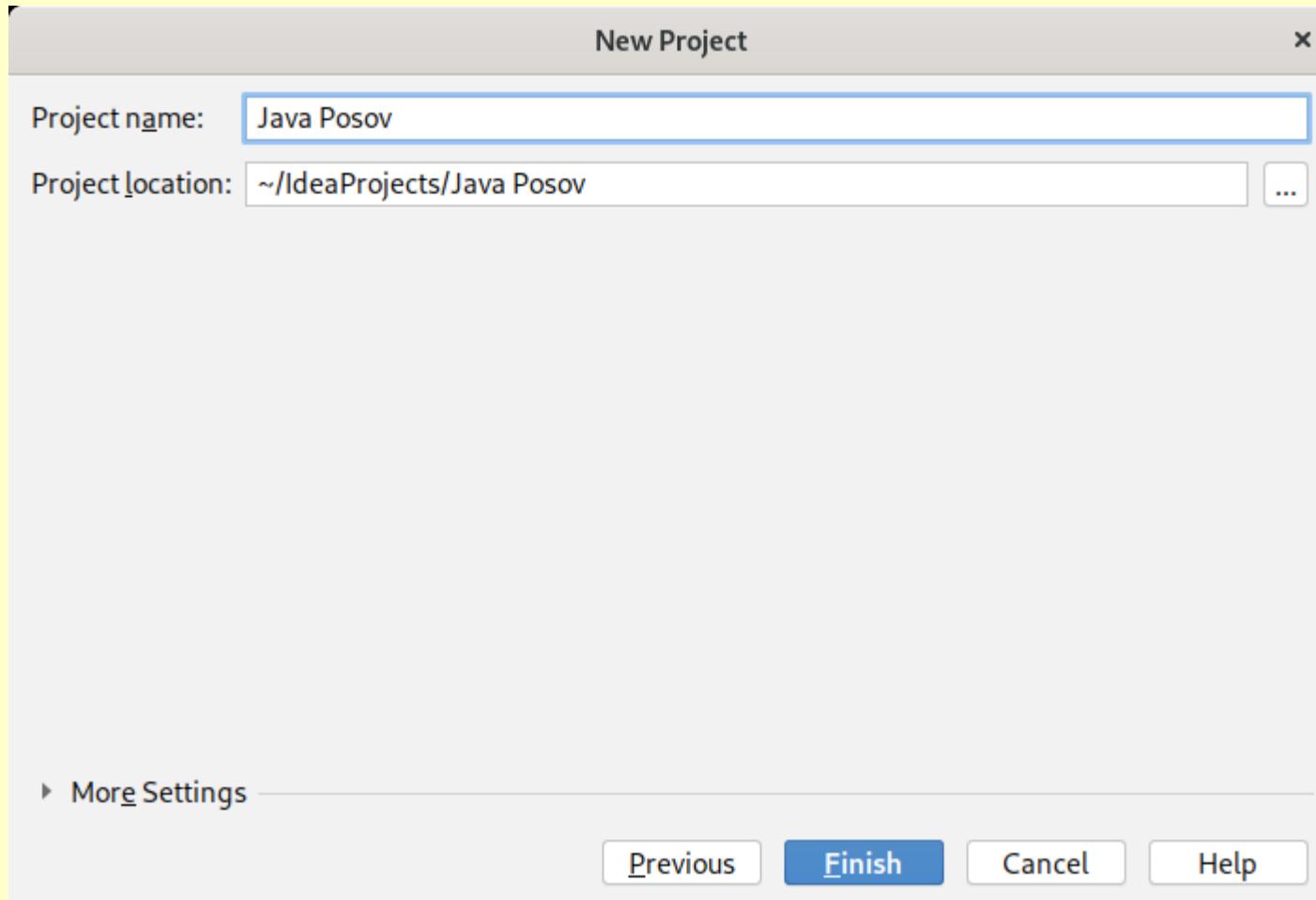


Нажмите Create New Project для создания нового проекта. Если у вас не будет начального окна, как сейчас, создать новый проект всегда можно через меню File.

Первым делом вы должны выбрать Java (JDK), которой будете пользоваться. Вспомните, что мы ее скачивали и разархивировали в какой-то каталог, который вы должны были запомнить. Если не запомнили, ищите или скачивайте JDK еще раз. Ниже видно, куда нажать (Add JDK), чтобы добавить свой JDK, если его нет в списке:



Пропускайте экраны, пока не увидите самый важный экран создания проекта про его расположение и название:



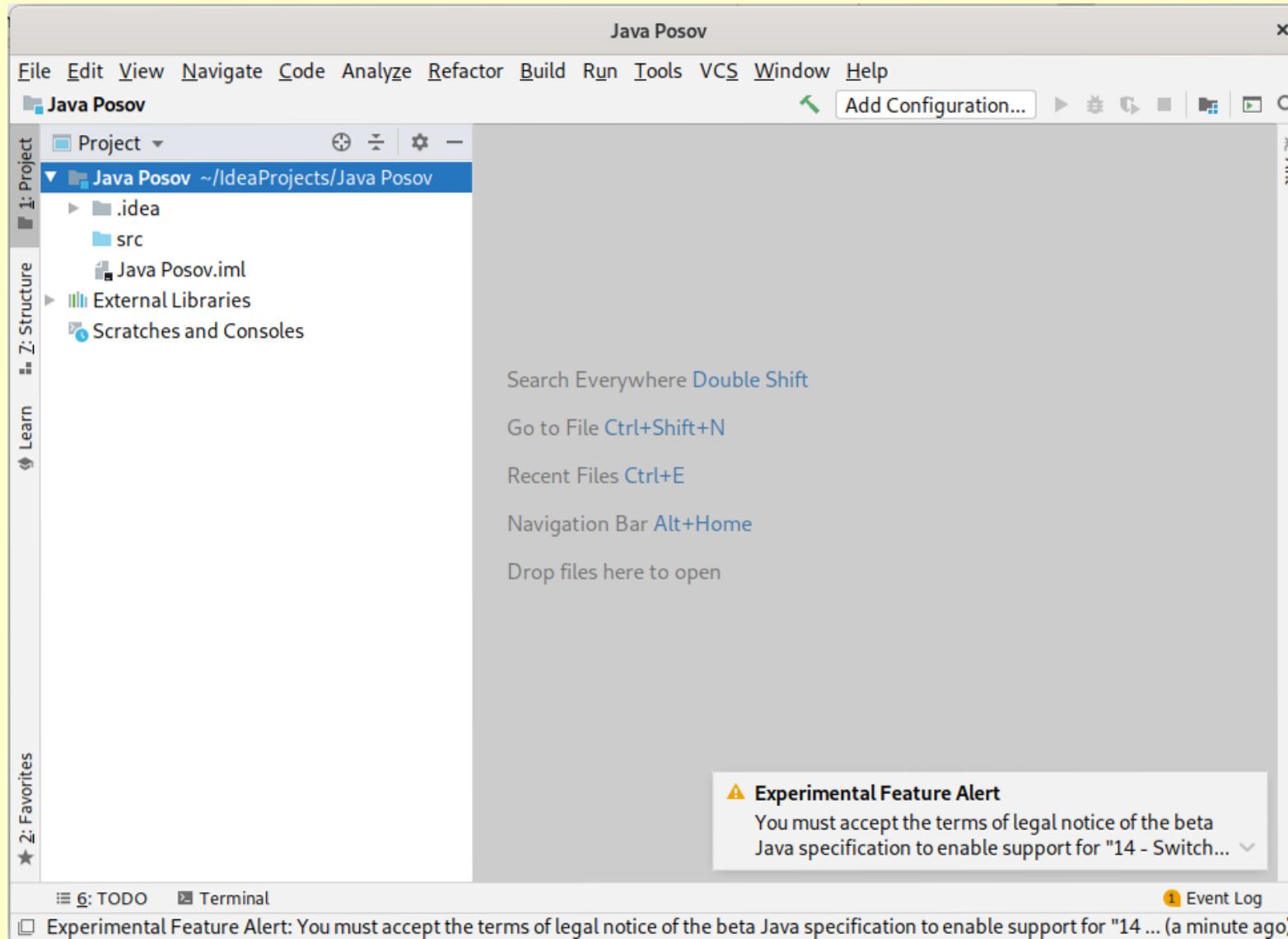
Придумайте название проекту. Там вы можете написать номер семестра, слово Java, еще какие-то логичные слова. После этого выбирайте расположение. Это папка, в которой будет находиться ваш проект. Важно:

Вы должны сознательно выбрать эту папку и знать, где она находится. Потом вам потребуется находить ее на диске, копировать куда-нибудь для сохранности, архивировать, чтобы отправить преподавателю и т.п.

Либо создайте новую папку, либо выберите папку, в которой вы решали задачи. Там у вас java и bat файлы.

Если вы когда-нибудь в будущем будете открывать свой проект, выбирайте для открытия ровно ту же папку, которую вы указали при создании проекта. Это очень частая ошибка, при открытии проекта указать какую-то подпапку, проект при этом открывается, но выглядит странно и не работает.

Нажимайте Finish и встречайте свой новый проект:



Слева видны файлы проекта. Если не видны, нажмите слева на кнопку “1: Project” или нажмите Alt + 1.

Ведите программу

```
public class HelloWorld {  
    public static void main(String[] args){  
        System.out.println("Hello World!");  
    }  
}
```

и нажмите зеленый треугольник слева от метода main. Нужен первый вариант, Run HelloWorld.main().

После первого запуска программы вы увидите снизу результат запуска, и еще один зеленый треугольник слева, который тоже позволяет запускать программу.

The screenshot shows the Java Posov IDE interface. The top bar includes File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, and Help menus. The title bar says "Java Posov - HelloWorld.java". The left sidebar has tabs for Project (selected), Structure, Learn, and Favorites. The main editor area displays the following Java code:

```
1 public class HelloWorld {  
2     public static void main(String[] args){  
3         System.out.println("Hello World!");  
4     }  
5 }
```

Below the code, the Run tab is selected, showing the command: "/home/ilya/opt/jdk14/bin/java -javaagent:/home/ilya/.local/share/Java Posov/src>HelloWorld". The output window shows the result: "Hello World!". At the bottom, status information includes "Process finished with exit code 0", "Build completed successfully in 6 s 854 ... (moments ago)", and file details like 2:7 LF, UTF-8, 4 spaces.

Создание AVD. Структура Android-проекта.

Для того, чтобы тестировать приложения, нам понадобится Android Virtual Device (AVD). Это эмулятор Android-смартфона, на который мы сможем устанавливать созданные нами приложения, и запускать их там.

Давайте его создадим под конкретное мобильное устройство. Нажимаем AVD Manager, далее +Creator Virtual Device, выбираем тип устройства, затем модель. Нажимаем кнопку Next.

Выбираем уровень API и версию Android. Нажимаем соответствующий Download и после загрузки необходимых компонент нажимаем Finish. AVD создан и настроен под конкретное мобильное устройство.

В предыдущем материале мы установили среду разработки и Android SDK.

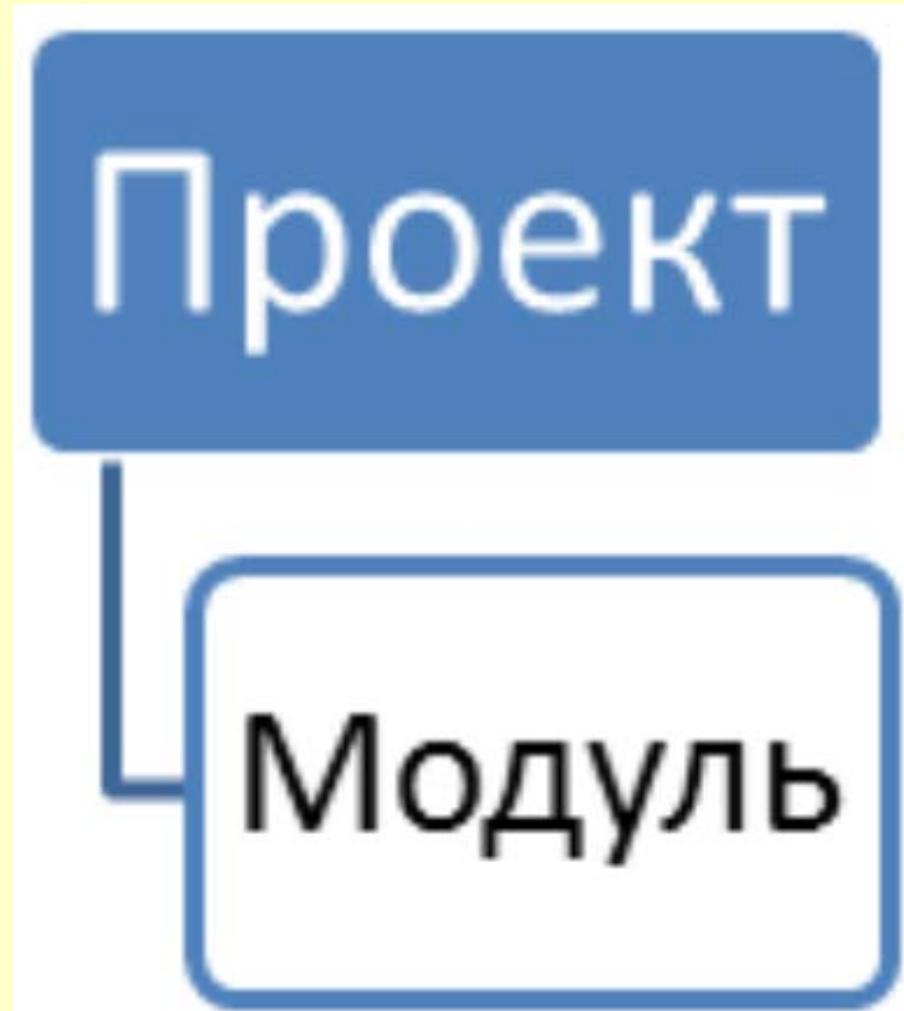
Теперь наконец-то мы можем создать наше первое приложение и посмотреть, как оно работает.

Чтобы создать приложение, нам нужно в Android Studio создать проект. При создании проекта, в нем создается модуль.

В этом модуле мы рисуем экраны приложения и пишем код. И при запуске этого модуля мы получаем готовое приложение.

Поэтому модуль по сути и является приложением. А проект - контейнер для модуля.

Т.е. в самом простом случае структура проекта такова:



Есть проект, и в нем есть модуль.

При запуске проекта запускается модуль и мы получаем Android-приложение, которое создано в этом модуле.

В этом случае: один проект = одно Android-приложение (один модуль).

Но в одном проекте может быть несколько модулей. Да и проектов можно создать несколько.



Здесь в первом проекте созданы два модуля, а во втором проекте – три модуля.

При запуске какого-либо проекта необходимо будет указать, какой именно модуль вы хотите запустить. И каждый модуль является отдельным Android-приложением.

Т.е. в этом случае: один проект = несколько Android-приложений (несколько модулей).

Пока не будем вдаваться в детали, какая из предложенных схем лучше и удобнее.

Для прохождения материала можно создать один проект, и в нем создавать модули для каждого занятия.

Либо можно создавать отдельный проект, например, на каждые 10 занятий. Можно вообще создавать отдельный проект на каждое занятие.

Мы начнем с варианта: один проект под все занятия. А со временем, как освоитесь, сами решите, какой вариант вам удобнее.

Создание проекта

Проект Android включает в себя все файлы, которые содержат исходный код приложения.

В данном материале рассказывается о двух способах создания нового проекта: в Android Studio либо в командной строке, используя инструментарий SDK.

Создание проекта в Android Studio

1. Создайте новый проект в Android Studio:

- Щелкните New Project на экране приветствия.
- Если открыт другой проект, щелкните меню File и выберите New Project.

2. Заполните поля в окне *Configure your new project*. Будет проще, если вы введете значения, как указано на рисунке 1.

- **Name (Application name)** это название приложения, которое увидят пользователи. Для текущего проекта напишите “My First App”.
- **Company domain** это название пространства имен, которое будет добавлено к названию пакета. Android Studio сохранит это значение для всех будущих проектов.
- **Package name** это полное наименование пакета проекта (согласно правилам именования пакетов в языке Java). Название пакета должно быть уникальным и не может совпадать с названиями других пакетов, установленных на устройстве. Вы можете использовать любое название пакета, независимо от названия приложения или пространства имен.
- **Save Location (Project location)** это директория на вашем компьютере в которой хранятся все файлы проекта.

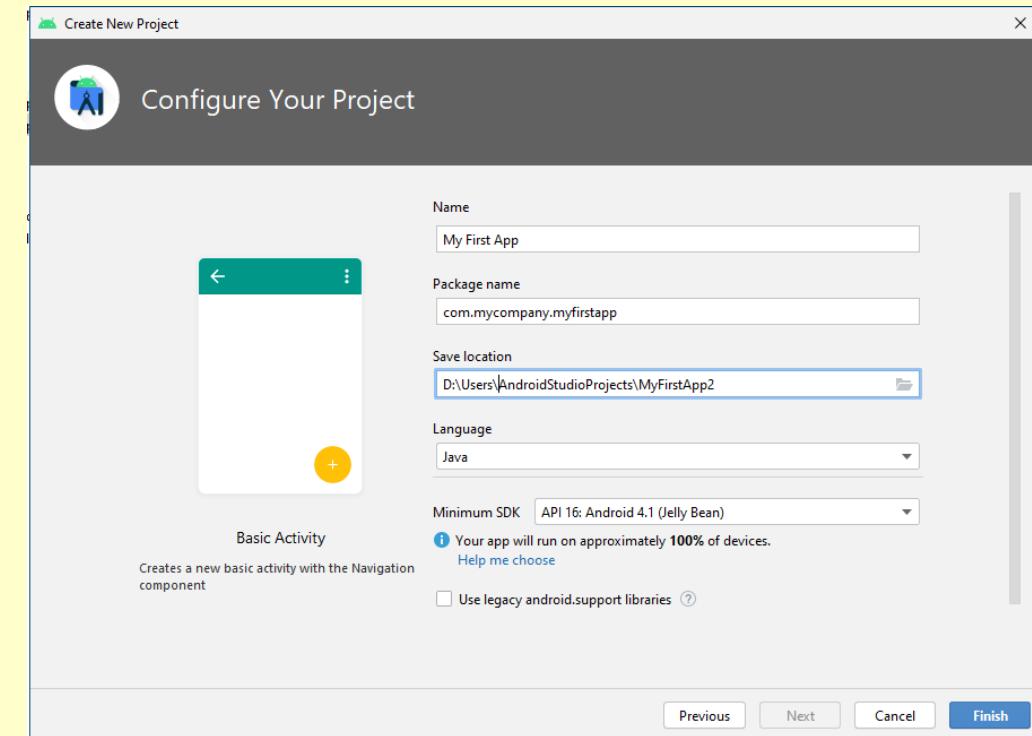


Рисунок 1. Создание нового проекта в Android Studio

3. В разделе Select the form factors your app will run on (выберите типы устройств, на которых можно запускать приложение), установите галочку Phone and Tablet (Смартфоны и планшеты).
4. Выберите API xx: Android y.y в поле Minimum SDK. Минимально необходимый SDK – это младшая версия Android, на котором ваше приложение сможет работать. Для указания версии используются уровни API. Для поддержки как можно большего числа устройств, вы должны выбрать наименьшую версию API, при которой ваше приложение выполняет свой основной набор функций. Если какие-либо некритичные функции вашего приложения работают только на новых версиях Android, вы можете включать их только при запуске на устройствах с определенной версией
5. Уберите галочки со всех других параметров (TV, Wear, Glass) и нажмите Next.
6. В окне Add an activity to your project (добавить явление в проект), выберите Blank Activity (пустое явление) и нажмите Next.

Что такое Явление (Activity)

Явление – это одна из отличительных особенностей Android фреймворка. Явления предоставляют пользователю доступ к вашему приложению. Приложение может содержать несколько явлений. Обычно приложение содержит главное явление, которое отображается при запуске и прочие явления, для выполнения разных задач в рамках приложения, например для просмотра документов. Можно воспринимать явления как отдельный экран приложения, аналог окна в других операционных системах, включающий в себя элементы интерфейса и программный код.

7. В окне *Describe the new activity for your project*, оставьте поля как они есть и нажмите *Finish*

Будет создан новый проект, содержащий некоторые стандартные файлы. Рассмотрим наиболее важные из них:

`app/src/main/res/layout/activity_my.xml`

Это XML файл разметки явления мы добавили при создании проекта в Android Studio. После создания нового проекта, Android Studio показывает этот файл в текстовом виде и в режиме предпросмотра экрана устройства. Файл по умолчанию содержит элемент `TextView`, который отображает строку “Hello world!”

`app/src/main/java/com.mycompany.myfirstapp/MyActivity.java`

Закладка с этим файлом появляется в Android Studio сразу после создания проекта. Файл содержит описание класса созданного явления. Если вы скомпилируете и запустите приложение, класс `Activity` загрузит файл разметки и отобразит надпись “Hello, world!”

`app/src/res/AndroidManifest.xml`

Файл манифеста описывает основные характеристики приложения и всех его компонентов. Мы вернемся к этому файлу позже, когда будем добавлять компоненты в приложение.

app/build.gradle

Android Studio использует Gradle для компиляции и сборки приложений. Существуют файлы build.gradle как для каждого модуля, так и для всего приложения в целом. Обычно вы будете иметь дело с файлами для модулей, в данном примере для модуля app. В файле указываются зависимости приложения, а так же стандартные настройки:

- `compiledSdkVersion` – версия платформы, на которой будет производиться компиляция приложения. По умолчанию это последняя версия Android, установленная в вашем SDK. (Вы должны использовать Android 4.1 или выше. Установите нужную версию, используя SDK Manager). Вы все еще можете создавать приложения с поддержкой старых версий, однако выбор последней версии для сборки приложения позволяет использовать новые возможности и оптимизировать работу приложения для свежих устройств.
- `applicationId` – полное наименование пакета вашего приложения, которое вы указали при создании проекта.
- `minSdkVersion` – минимально необходимая версия андроид, которую вы указали при создании проекта. Это младшая версия андроид, которую поддерживает ваше приложение.
- `targetSdkVersion` – указывает на самую большую версию андроид, на которой вы проверяли работу вашего приложения. Рекомендуем тестировать приложение при выходе каждой новой версии андроид и увеличивать значение `targetSdkVersion`. Таким образом вам будут доступны новые возможности платформы.

Также отметим содержимое директории `/res`, которая содержит ресурсы приложения:

`drawable-hdpi/`

Содержит нарисованные объекты, такие как растровые изображения, для экранов высокого разрешения (`hdpi`). В других директориях `drawable` содержатся изображения для экранов с разным разрешением. К примеру в ней находится файл `ic_launcher.png`, содержащий иконку приложения.

`layout/`

Содержит файлы разметки пользовательского интерфейса. К примеру файл `activity_my.xml`, о котором говорилось выше, содержит разметку для класса `MyActivity`.

`values/`

Содержит XML файлы, включающие в себя такие ресурсы, как строки и определение цветов. К примеру в файле `string.xml` хранится строка “Hello world!”, отображаемая при запуске приложения.

Создание проекта с использованием командной строки

Если вы не используете среду разработки Android Studio, вы можете создать проект используя командную строку:

1. Перейдите в директорию tools/ вашего Android SDK.
2. Выполните команду:

```
android list targets
```

Данная команда выводит список доступных в вашем SDK платформ. Найдите платформу, на которой вы хотите выполнять компиляцию вашего приложения и запомните targetID. Мы рекомендуем выбирать максимально доступную версию. Вы все еще можете создавать приложения с поддержкой старых версий, однако выбор последней версии для сборки приложения позволяет использовать новые возможности и оптимизировать работу приложения для свежих устройств.

Если список доступных платформ пуст, вам необходимо их установить, используя Android SDK Manager.

3. Выполните команду:

```
android create project --target <target-id> --name MyFirstApp \ <br>
--path <path-to-workspace>/MyFirstApp --activity MyActivity \ <br>
--package com.example.myfirstapp <br>
```

Где <target-id> замените на идентификатор из списка доступных платформ, а вместо <path-to-workspace> укажите директорию для хранения файлов проекта.

Совет: добавьте директории platform-tools/ и tools/ в переменную окружения PATH.

Запуск вашего приложения

- Если вы следовали инструкциям предудыщего материала, то сейчас у вас есть все необходимое для немедленного запуска приложения.
- Как вы будете запускать приложение зависит от двух вещей: имеется ли у вас реальное Android устройство и используете ли вы Android Studio.

Запуск на реальном устройстве

Если у вас есть устройство, работающее под Android, следуйте инструкциям ниже для установки и запуска приложения.

Подготовка вашего устройства

1. Подключите ваше устройство к компьютеру с помощью USB кабеля. Если вы используете операционную систему Windows, может понадобиться установка USB драйвера для вашего устройства. Подробнее об установке драйверов читайте в разделе OEM USB драйвера.
2. Включите на устройстве режим USB отладки.
 - На большинстве устройств, работающих под управлением Android 3.2 и старше данная опция находится в меню Настройки > Приложения > Разработка
 - В андроид 4.0 и новее опция находится в меню Настройка > Разработка.

Примечание: В Android 4.2 и новее, пункт меню Разработка по умолчанию скрыт. Чтобы отобразить его в меню, нажмите Настройки > О телефоне и щелкните по пункту Номер сборки семь раз. После этого вернитесь в предыдущее меню и найдите пункт Разработка.

Запуск приложения из Android Studio

1. Выберите один из файлов вашего проекта и нажмите кнопку Run на панели инструментов.
2. В появившемся окне Choose Device (выбор устройства), активируйте пункт Choose a running device (выбрать подключенное устройство) и выберите ваш телефон. Нажмите OK.

Android Studio установит приложение на подключенное устройство и запустит его.

Запуск приложения из командной строки

1. Перейдите в корневую директорию вашего проекта и выполните команду

```
ant debug
```

2. Убедитесь, что директория platform-tools/ добавлена в переменную окружения PATH и выполните команду:

```
adb install bin/MyFirstApp-debug.apk
```

Найдите на вашем устройстве приложение MyFirstApp и запустите его.

Как видите, собрать и запустить приложение на вашем устройстве очень просто!

Запуск приложения в эмуляторе

1. Запустить менеджер виртуальных устройств можно двумя способами:

В Android Studio выберите Tools > Android > AVD Manager или щелкните по иконке менеджера на панели инструментов.

В командной строке перейдите в директорию <sdk>/tools/ и выполните команду:

```
android avd
```

Примечание: Внешний вид менеджера виртуальных устройств, запущенного из командной строки отличается от менеджера в Android Studio, поэтому действия, показанные ниже могут отличаться.



Рисунок 2. Менеджер виртуальных устройств

2. В окне менеджера устройств щелкните Create Virtual Device.
3. В следующем окне выберите конфигурацию подходящего устройства, например Nexus 6 и нажмите Next.
4. Выберите желаемую версию AVD и нажмите Next.
5. Проверьте правильность заполнения конфигурации и нажмите Finish.

Информацию об использовании AVD рассмотрим позднее

Запуск приложения из Android Studio

1. Выберите один из файлов вашего проекта и нажмите кнопку Run на панели инструментов.
2. В появившемся окне Choose Device (выбор устройства), активируйте пункт Launch emulator (выбрать эмулятор)
3. В списке виртуальных устройств выберите созданный вами эмулятор и нажмите OK.

Запуск эмулятора может занять несколько минут. После разблокировки экрана вы увидите запущенное приложение.

Запуск приложения из командной строки

1. Перейдите в корневую директорию вашего проекта и выполните команду

`ant debug`

2. Убедитесь, что директория `platform-tools/` добавлена в переменную окружения PATH и выполните команду:

`adb install bin/MyFirstApp-debug.apk`

Найдите в эмуляторе приложение MyFirstApp и запустите его.

Создание простого интерфейса пользователя

В данном материале мы создадим XML разметку, включающую текстовое поле и кнопку. В следующем материале мы научимся обрабатывать нажатие на кнопку и передавать данные из текстового поля в другое явление.

Графический интерфейс пользователя в Android строится на основе дерева объектов типа `View` и `ViewGroup`. Объекты типа `View` – это обычные виджеты, такие как кнопки и поля ввода. Объекты типа `ViewGroup` это невидимые контейнеры, которые управляют расположением вложенных элементов, например в виде таблицы или вертикального списка.

Android предоставляет набор XML элементов, соответствующих наследникам `View` и `ViewGroup`, чтобы вы могли описывать интерфейс пользователя с помощью XML.

Объекты разметки (`Layouts`) являются наследниками класса `ViewGroup`. В данном материале мы будем работать с линейной разметкой `LinearLayout`.

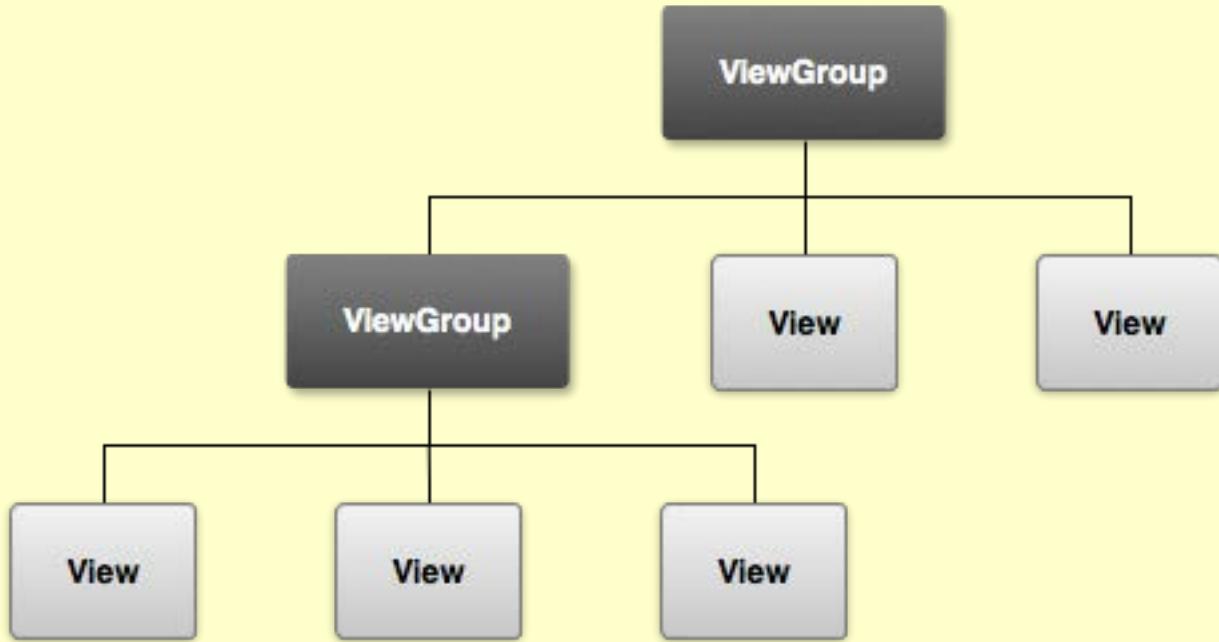


Рисунок 3 .ViewGroup позволяет строить дерево объектов разметки.

Различная разметка

Создание разметки в XML файлах предпочтительнее, чем в исходном коде по нескольким причинам, но главным образом из-за необходимости создания различных файлов разметки для устройств с различными размерами экрана.

К примеру, вы можете создать две версии разметки и заставить систему отображать одну из них на маленьких экранах, а другую на больших.

Создание линейной разметки (Linear Layout)

1. В Android Studio откройте файл res/layout/activity_my.xml

Шаблон BlankActivity, который вы выбрали при создании проекта включает в себя файл activity_my.xml, который содержит корневой элемент RelativeLayout с вложенным виджетом TextView.

3. В окне предпросмотра щелкните по иконке Hide , чтобы скрыть окно.

В Android Studio при открытии файла разметки по умолчанию открывается окно предпросмотра, которое содержит WYSIWYG редактор для создания интерфейса пользователя. Однако в данном материале мы будем работать напрямую с XML файлом.

3. Удалите элемент <TextView>.

4. Измените элемент <RelativeLayout> на <LinearLayout>.

5. Добавьте атрибут android:orientation и установите для него значение "horizontal".

6. Удалите атрибуты android:padding и tools:context.

В результате разметка должна выглядеть так:

res/layout/activity_my.xml

```
1 <LinearLayout xmlns:android="schemas.android.com/apk/res/android"  
2     xmlns:tools="schemas.android.com/tools"  
3     android:layout_width="match_parent"  
4     android:layout_height="match_parent"  
5     android:orientation="horizontal">  
6 </LinearLayout>
```

LinearLayout является наследником класса ViewGroup и размещает вложенные элементы в вертикальном или горизонтальном порядке, в зависимости от значения атрибута android:orientation. Вложенные в LinearLayout элементы отображаются на экране в том порядке, в котором они описаны в XML.

Все элементы разметки должны иметь атрибуты android:layout_width и android:layout_height, которые указывают на их размер.

Поскольку LinearLayout корневой элемент разметки, он должен быть растянут на весь экран. Для этого укажем значение атрибутов android:layout_width и android:layout_height равное "match_parent". Это означает, что ширина и высота элемента должна соответствовать ширине и высоте родительского элемента.

Добавление текстового поля

Как и для каждого объекта типа View, необходимо указать некоторые XML атрибуты, характерные для элемента EditText.

1. В файле activity_my.xml, создайте внутри <LinearLayout> элемент <EditText> и укажите для него атрибут android:id со значением @+id/edit_message.

2. Создайте атрибуты layout_width и layout_height со значением "wrap_content".

3. Создайте атрибут hint и укажите в качестве значения строковый объект с названием edit_message.

В результате элемент <EditText> должен выглядеть следующим образом:
res/layout/activity_my.xml

```
1 <EditText android:id="@+id/edit_message"  
2     android:layout_width="wrap_content"  
3     android:layout_height="wrap_content"  
4     android:hint="@string/edit_message" />
```

Атрибуты элемента <EditText> означают следующее:

android:id

Этот атрибут задает уникальный идентификатор элемента, по которому вы можете ссылаться на объект из программного кода и работать с ним. (Как это делается вы узнаете в следующем уроке). Значок "@" требуется указывать, если вы ссылаетесь на какой-либо объект ресурса из XML. После значка идет тип ресурса (в данном примере id), затем имя ресурса после слэша (edit_message).

Значок "+" перед типом ресурса необходим только при первоначальном указании идентификатора. При компиляции приложения, SDK использует указанный идентификатор для создания переменной в файле gen/R.java. Эта переменная будет указывать на элемент EditText. После этого в указании значка "+" нет необходимости. Другими словами, значок "+" необходимо указывать только при создании нового идентификатора. Нет необходимости добавлять "+" при работе с существующими ресурсами, такими как строки или разметка.

android:layout_width и android:layout_height

Вместо указания конкретных размеров элемента, мы указали "wrap_content", что позволяет элементу менять размер в зависимости от содержимого. Если мы зададим значение "match_parent", то элемент EditText заполнит весь экран, поскольку его ширина и высота будут равны родительскому LinearLayout.

android:hint

Этот атрибут задает подсказку, которая отображается в текстовом поле, если оно не заполнено.

Вместо жестко заданной строки мы указали ссылку на строковый ресурс "@string/edit_message", который находится в другом файле. Поскольку это ссылка на конкретный ресурс, нет необходимости добавлять значок "+". Однако поскольку вы еще не создали данный строковый ресурс, компилятор выдаст ошибку. Мы добавим ресурс немного позже.

Примечание: Указанный строковый ресурс имеет такое же имя, как и идентификатор элемента. Однако при ссылке на ресурс всегда учитывается его тип (например id или string), поэтому использование одинаковых имен не вызывает проблем.

Объекты ресурсов

Объект ресурса имеет уникальное целочисленное значение, связанное с ресурсами приложения, такими как растровые изображения, файлы разметки или строки.

Каждый ресурс соответствует объекту ресурса, объявленному в файле gen/R.java.

Вы можете использовать имя объекта из класса R для ссылки на ресурсы из кода, например если вам нужно программно изменить атрибут android:hint.

В качестве атрибута android:id вы можете использовать произвольные наименования.

SDK автоматически генерирует файл R.java при каждой компиляции приложения, поэтому вы не должны вручную изменять этот файл.

Создание строкового ресурса

По умолчанию строковые ресурсы хранятся в файле res/values/strings.xml. Добавим новый ресурс "edit_message" и укажем для него значение “Ведите сообщение”.

1. В Android Studio откройте файл res/values/strings.xml
2. Добавьте строку с названием "edit_message" и значением “Ведите сообщение”.
3. Добавьте строку с названием "button_send" и значением “Отправить”. Позже мы создадим кнопку, которая будет использовать данный ресурс.
4. Удалите строку, содержащую надпись "hello world".

Теперь файл strings.xml должен выглядеть следующим образом:

res/values/strings.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3     <string name="app_name">Мое первое приложение</string>
4     <string name="edit_message">Введите сообщение</string>
5     <string name="button_send">Отправить</string>
6     <string name="action_settings">Настройки</string>
7     <string name="title_activity_main">MainActivity</string>
8 </resources>
```

Для надписей пользовательского интерфейса всегда используйте строковые ресурсы. Строковые ресурсы позволяют управлять надписями в одном месте, делают их проще для поиска и редактирования.

Более того, строковые ресурсы позволяют вам с легкостью перевести приложение на разные языки.

Добавление кнопки

1. В Android Studio откройте файл res/layout/activity_my.xml.
2. Создайте внутри <LinearLayout> элемент <Button> сразу после элемента <EditText>.
3. Задайте ширину как "wrap_content", чтобы она зависела от надписи внутри кнопки.
4. Добавьте атрибут android:text и укажите в качестве значения строковый ресурс "button_send", который мы создали в предыдущем параграфе.

Теперь <LinearLayout> должен выглядеть так:

res/layout/activity_my.xml

```
1 <LinearLayout xmlns:android="schemas.android.com/apk/res/android"
2     xmlns:tools="schemas.android.com/tools"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent"
5     android:orientation="horizontal" >
6     <EditText android:id="@+id/edit_message"
7         android:layout_width="wrap_content"
8         android:layout_height="wrap_content"
9         android:hint="@string/edit_message" />
10    <Button
11        android:layout_width="wrap_content"
12        android:layout_height="wrap_content"
13        android:text="@string/button_send" />
14 </LinearLayout>
```

Примечание: Мы не указываем для кнопки атрибут android:id, поскольку не собираемся обращаться к ней из кода.

Мы создали разметку, в которой оба виджета EditText и Button имеют размеры, соответствующие их содержимому. Смотрите рисунок 2

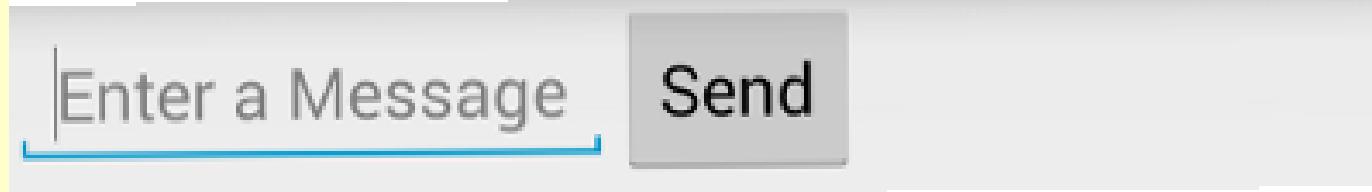


Рисунок 2.Кнопка и поле вводе с шириной равной "wrap_content".

Это удобно для кнопки, но плохо для текстового поля, потому что пользователь может вводить довольно длинную строку. Было бы неплохо растянуть текстовое поле на все неиспользуемое пространство экрана. При использовании LinearLayout это можно сделать, добавив атрибут веса android:layout_weight.

Вес – это число, показывающее количество пространства, которое может использовать каждый из элементов относительно друг друга. Это как в рецепте коктейля: “2 части водки, 1 часть сока” – это означает, что две трети коктейля состоит из водки. Рассмотрим пример. Если вы указываете для одного элемента значение 2, а для другого 1, сумма будет равна трем и первый элемент займет 2/3 свободного пространства, а второй заполнит остаток. Если вы добавите третий элемент и укажете для него вес, равный 1, то первый элемент (с весом 2) займет 1/2 пространства, а оставшиеся два элемента займут по 1/4.

Стандартный вес для всех элементов равен 0. Если вы укажете значение больше нуля только для одного элемента, то данный элемент заполнит все пространство, оставшееся после прорисовки других элементов.

Растягиваем поле ввода

Для того, чтобы растянуть элемент EditText на все свободное пространство проделаем следующее:

В файле activity_my.xml добавим элементу <EditText> атрибут layout_weight со значением 1, а атрибуту layout_width установим значение 0dp.

res/layout/activity_my.xml

```
1 <EditText  
2     android:layout_weight="1"  
3     android:layout_width="0dp"  
4     ... />
```

Мы указали ширину EditText равную 0dp. Установка нулевой ширины улучшает производительность разметки.

Использование "wrap_content" требует от системы лишнего вычисления ширины, результат которого в конечном счете не имеет смысла, поскольку указание веса все равно запустит операцию расчета свободного пространства.

На рисунке 3 показан результат применения весового коэффициента к элементу EditText.

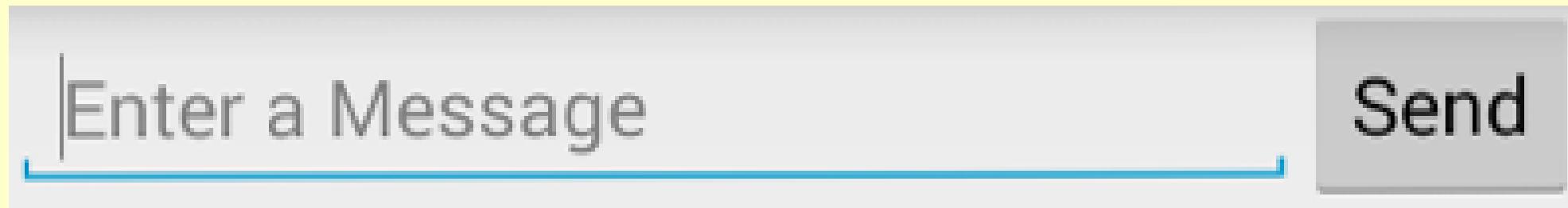


Рисунок 3. Вид поля ввода при установленном параметре weight.

Законченный вариант файла activity_my.xml выглядит таким образом:

res/layout/activity_my.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="schemas.android.com/apk/res/android"
3   xmlns:tools="schemas.android.com/tools"
4   android:layout_width="match_parent"
5   android:layout_height="match_parent"
6   android:orientation="horizontal">
7   <EditText android:id="@+id/edit_message"
8     android:layout_weight="1"
9     android:layout_width="0dp"
10    android:layout_height="wrap_content"
11    android:hint="@string/edit_message" />
12   <Button
13     android:layout_width="wrap_content"
14     android:layout_height="wrap_content"
15     android:text="@string/button_send" />
16 </LinearLayout>
```

Запуск приложения

Созданная разметка используется классом Activity, который был сгенерирован SDK при создании проекта. Запустите приложение, чтобы посмотреть результат:

1. В Android Studio щелкните по кнопке Run на панели инструментов.
2. Если вы пользуетесь командной строкой, перейдите в корневую директорию вашего проекта и выполните команды:

```
1 ant debug  
2 adb install bin/MyFirstApp-debug.apk
```

Запуск другого явления

После завершения предыдущего материала, у нас есть приложение, отображающее единственное явление, которое содержит текстовое поле ввода и кнопку. В данном занятии мы напишем код, который будет открывать новое явление при нажатии на кнопку.

Обработка нажатия на кнопку

Обработка нажатия на кнопку

1. В Android Studio откройте файл res/layout/activity_my.xml
2. Добавьте элементу <Button> атрибут android:onClick

res/layout/activity_my.xml

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/button_send"  
    android:onClick="sendMessage" />
```

Значение "sendMessage" атрибута android:onClick, это название метода в MyActivity, который будет вызван при нажатии на кнопку.

3. Откройте файл java/com.mycompany.myfirstapp/MyActivity.java

4. Добавьте метод sendMessage в класс MyActivity, как показано ниже

java/com.mycompany.myfirstapp/MyActivity.java

```
/** Данный метод вызывается при нажатии на кнопку */
```

```
public void sendMessage(View view) {
```

```
    // Обработка реакции нажатия
```

```
}
```

Для того, чтобы система сопоставила данный метод и имя метода, указанное в атрибуте android:onClick, метод должен удовлетворять следующим условиям:

- Быть публичным (public).
- Возвращать тип void.
- Иметь идентичный параметр типа View (в параметр передается объект View, который вызвал метод).

Далее мы напишем тело метода для передачи текста из поля ввода в другое явление.

Создание намерения (Intent)

1. В файле MyActivity.java создайте объект намерения типа Intent внутри метода sendMessage(). С помощью данного намерения мы будем запускать явление под названием DisplayMessageActivity.

java/com.mycompany.myfirstapp/MyActivity.java

```
public void sendMessage(View view) {  
    Intent intent = new Intent(this, DisplayMessageActivity.class);  
}
```

Примечание: Ссылка на DisplayMessageActivity вызовет ошибку в Android Studio, поскольку мы еще не создали класс с таким именем. Не беспокойтесь, скоро мы это сделаем.

Конструктор принимает два параметра:

- Первый параметр типа Context. Мы можем использовать this, поскольку класс Activity является наследником класса Context.
- Второй параметр типа Class указывает на компонент, к которому будет обращено данное намерение (В нашем случае явление, которое должно быть запущено).

Что такое намерение (Intents)

Намерение это объект, который выполняет связывание двух различных компонентов (например двух явлений). Намерения применяются для широкого спектра задач, но наиболее часто вы будете их использовать для запуска явлений. Подробную информацию смотрите в разделе Явления и фильтры.

2. Добавьте в начало файла импорт класса Intent:

```
java/com.mycompany.myfirstapp/MyActivity.java  
import android.content.Intent;
```

Совет: В Android Studio нажмите Alt+Enter (option + return на Mac), чтобы импортировать недостающие классы.

3. Внутри sendMessage() создайте объект типа EditText, и воспользуйтесь методом findViewById() для получения ссылки на текстовое поле из нашей разметки.

java/com.mycompany.myfirstapp/MyActivity.java

```
public void sendMessage(View view) {  
    Intent intent = new Intent(this, DisplayMessageActivity.class);  
    EditText editText = (EditText) findViewById(R.id.edit_message);  
}
```

4. Добавьте в начало файла импорт класса EditText.

5. Локальной переменной message присвойте значение из текстового поля и передайте это значение в намерение при помощи метода putExtra():

java/com.mycompany.myfirstapp/MyActivity.java

```
public void sendMessage(View view) {  
    Intent intent = new Intent(this, DisplayMessageActivity.class);  
    EditText editText = (EditText) findViewById(R.id.edit_message);  
    String message = editText.getText().toString();  
    intent.putExtra(EXTRA_MESSAGE, message);  
}
```

Класс Intent позволяет передавать пары ключ-значение, называемые дополнительными данными.

Метод putExtra() принимает первым параметром ключ, а вторым параметром значение.

6. Объявите внутри класса MyActivity переменную EXTRA_MESSAGE следующим образом:

java/com.mycompany.myfirstapp/MyActivity.java

```
public class MyActivity extends ActionBarActivity {
```

```
    public final static String EXTRA_MESSAGE =  
"com.mycompany.myfirstapp.MESSAGE";
```

```
...
```

```
}
```

Для того, чтобы другое явление смогло запросить дополнительные данные, необходимо сделать переменную с названием ключа публичной константой.

В целом использование названия пакета при объявлении ключей дополнительных данных, является хорошим стилем и мы рекомендуем его придерживаться.

Это гарантирует, что ключ будет уникальным при взаимодействии вашего приложения с любыми другими приложениями.

7. Внутри метода sendMessage() добавьте вызов startActivity() и передайте в качестве параметра созданный объект Intent.

Законченный метод sendMessage(), вызываемый при нажатии кнопки теперь выглядит так:

```
java/com.mycompany.myfirstapp/MyActivity.java
```

```
/** Вызывается при нажатии на кнопку */
```

```
public void sendMessage(View view) {
```

```
    Intent intent = new Intent(this, DisplayMessageActivity.class);
```

```
    EditText editText = (EditText) findViewById(R.id.edit_message);
```

```
    String message = editText.getText().toString();
```

```
    intent.putExtra(EXTRA_MESSAGE, message);
```

```
    startActivity(intent);
```

```
}
```

Система обрабатывает вызов и запускает экземпляр класса Activity, заданный в объекте типа Intent. Теперь чтобы все заработало, нам нужно создать класс DisplayMessageActivity.

Создание второго явления

Все наследники класса Activity должны реализовывать метод onCreate().

В данном методе явление получает и обрабатывает данные от намерений и выполняет первоначальную настройку всех компонентов явления.

Также в методе onCreate() должен вызываться метод setContentView() для выбора файла разметки данного явления.

Создание нового явления в Android Studio

Android Studio автоматически добавляет заготовку метода `onCreate()` при создании нового явления.

1. В Android Studio щелкните правой кнопкой мыши по пакету `java/com.mycompany.myfirstapp` и выберите `New > Activity > Blank Activity`.
 2. В окне `Choose options` заполните поля:
 - **Activity Name (Название явления):** `DisplayMessageActivity`
 - **Layout Name (Название разметки):** `activity_display_message`
 - **Title (Заголовок):** Моё сообщение
 - **Hierarchical Parent (Родительский элемент):** `com.mycompany.myfirstapp.MyActivity`
 - **Package name (Название пакета):** `com.mycompany.myfirstapp`
- Нажмите `Finish`.

3. Откройте файл `DisplayMessageActivity.java`.

Как видите, обязательный метод `onCreate()` уже создан. Позже мы добавим в него нужный код. Также по умолчанию добавлен метод `onOptionsItemSelected()`, в котором описываются правила работы панели инструментов. Оставьте пока все как есть.

4. Удалите метод `onCreateOptionsMenu()`, в данном приложении он не понадобится.

Если вы работаете в `Android Studio`, то сейчас уже можете запустить обновленное приложение.

При нажатии на кнопку будет открываться второе явление, однако надпись в нем не будет меняться.

Дальше мы допишем наше новое явление, чтобы оно открывалось с текстом, введенным в поле ввода.

Создание явления без Android Studio

Для создания явления вы можете использовать любую другую среду разработки или командную строку.

1. Создайте новый файл DisplayMessageActivity.java в директории src/ вашего проекта.
2. Добавьте в файл следующий код:

```
public class DisplayMessageActivity extends ActionBarActivity {  
  
    @Override  
  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_display_message);  
        if (savedInstanceState == null) {  
            getSupportFragmentManager().beginTransaction()  
                .add(R.id.container, new PlaceholderFragment()).commit();  
        }  
    }  
}
```

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    int id = item.getItemId();  
    if (id == R.id.action_settings) {  
        return true;  
    }  
    return super.onOptionsItemSelected(item);  
}  
public static class PlaceholderFragment extends Fragment {  
    public PlaceholderFragment() {}  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup container,  
        Bundle savedInstanceState) {  
        View rootView = inflater.inflate(R.layout.fragment_display_message,  
            container, false);  
        return rootView;  
    }  
}
```

Примечание: Если вместо Android Studio вы используете какую-либо другую среду разработки, то в вашем проекте может не быть файла разметки activity_display_message, который передается первым параметром в setContentView().

Не волнуйтесь, позже мы исправим эту проблему.

3. Добавьте название нового явления в файл strings.xml:

```
<resources>
```

```
...
```

```
    <string name="title_activity_display_message">My Message</string>
```

```
</resources>
```

4. Добавьте в секцию <application> файла AndroidManifest.xml новый элемент <activity>. Это элемент для нашего класса DisplayMessageActivity.

```
<application ... >
```

```
...
```

```
<activity
```

```
    android:name="com.mycompany.myfirstapp.DisplayMessageActivity"
```

```
    android:label="@string/title_activity_display_message"
```

```
    android:parentActivityName="com.mycompany.myfirstapp.MyActivity" >
```

```
        <meta-data
```

```
            android:name="android.support.PARENT_ACTIVITY"
```

```
            android:value="com.mycompany.myfirstapp.MyActivity" />
```

```
    </activity>
```

```
</application>
```

Атрибут `android:parentActivityName` указывает на родительский элемент явления.

Система использует эти данные для установки правил навигации внутри приложения, таких как Иерархическая навигация в Android 4.1 (уровень API 16) и выше. Вы можете создать такую же навигацию и для старых версий Android, используя библиотеку поддержки и добавив элемент `<meta-data>` в `AndroidManifest.xml`.

Примечание: Если вы следовали инструкции по установке дополнительных пакетов SDK, ваш Android SDK уже включает в себя последнюю версию библиотеки поддержки. При использовании Android Studio, библиотека поддержки автоматически добавляется к вашему проекту (вы можете увидеть `.jar` файл в списке зависимостей). Если вы не используете Android Studio, вам нужно добавить файлы библиотеки вручную. Инструкцию вы можете посмотреть в разделе [Добавление библиотеки поддержки](#).

Если вы используете стороннюю среду разработки, не переживайте, что ваш проект не компилируется. Сейчас мы добавим код для отображения введенного текста.

Получение намерения

Любое явление запускается с помощью намерения, независимо от используемых элементов навигации.

Вы можете получить экземпляр объекта намерения Intent, запустивший явление с помощью метода getIntent().

Экземпляр Intent будет содержать также все дополнительные данные.

1. Откройте файл java/com.mycompany.myfirstapp/DisplayMessageActivity.java.
2. В методе onCreate() удалите следующую строку:

```
setContentView(R.layout.activity_display_message);
```

3. Создайте переменную типа Intent и присвойте ей значение getIntent().
4. В верхней части файла импортируйте класс Intent.

В Android Studio нажмите сочетание клавиш Alt+Enter (option + return на Mac) для импорта недостающих классов.

5. Получить дополнительные данные строкового типа, переданные из MyActivity, можно с помощью метода getStringExtra().

```
String message = intent.getStringExtra(MyActivity.EXTRA_MESSAGE);
```

Отображение сообщения

1. Создайте объект типа TextView в методе onCreate()

```
TextView textView = new TextView(this);
```

2. Установите размер шрифта и сообщение с помощью метода setText():

```
textView.setTextSize(40);
```

```
textView.setText(message);
```

3. Мы можем сделать объект типа TextView корневым элементом разметки явления. Для этого передадим объект в качестве параметра метода setContentView().

```
setContentView(textView);
```

4. Импортируйте класс TextView.

В Android Studio нажмите сочетание клавиш Alt+Enter (option + return на Mac) для импорта недостающих классов.

Теперь метод onCreate() класса DisplayMessageActivity должен выглядеть следующим образом:

```
@Override  
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    // Получить сообщение из Intent  
    Intent intent = getIntent();  
    String message = intent.getStringExtra(MyActivity.EXTRA_MESSAGE);  
    // Создание объекта TextView  
    TextView textView = new TextView(this);  
    textView.setTextSize(40);  
    textView.setText(message);  
    // Делаем textView корневым элементом разметки activity  
    setContentView(textView);  
}
```

Теперь вы можете запустить приложение. После запуска введите любой текст в поле ввода и нажмите кнопку Отправить.

Сообщение откроется во втором явлении.

Поздравляем! Вы создали свое первое приложение под Android!

Чтобы узнать больше о разработке для Android, начните изучение следующего материала Добавление панели инструментов.

Методические рекомендации по оформлению курсовой работы

Структура курсовой работы

1. Титульный лист.
2. Задание на курсовую.
3. Аннотация
4. Содержание
5. Основной текст(Отчет, пояснительная записка)
 - 5.1. Введение
 - 5.2. Разделы (от 1 до 3)
 - 5.2.1 Раздел анализа и исследований
 - 5.2.2 Раздел реализации проекта
 - 5.3. Заключение
 - 5.4. Список использованной литературы
 - 5.5. Приложения

Содержание

Содержание – вспомогательный элемент курсовой работы.

Здесь отображаются (с указанием номеров страниц) все структурные элементы в том же порядке и тех же формулировках, что и в тексте самой работы.

В содержание включаются:

- введение;
- заголовки глав и разделов;
- заключение;
- список использованной литературы;
- приложения.

Введение

Введение дает читателю начальное представление о сущности исследования. В нем обосновывается актуальность исследуемой проблемы, описываются поставленные цели, новизна, объект и предметная область исследования, практическая значимость.

Во введении перечисляются применяемые методы и способы исследования. Объем введения составляет не больше 5% от общего числа страниц.

Таким образом, если всего в вашей курсовой 20-40 страниц, то приемлемый объем введения составит 1,5-2 страницы.

Введение лучше писать после основного текста работы.

Основной текст(отчет, пояснительная записка)

Главный структурный элемент курсовой работы.

Здесь приводятся расчеты, представлены научные и практические изыскания студента.

Допустимый объем основного текста курсовой работы регламентируется внутренними документами образовательных организаций.

Обычно это 20 – 40 страниц.

Курсовая работа традиционно состоит из двух или трех глав или разделов.

Количество глав зависит от темы и типа курсовой работы.

Имеющая расчетно-практический характер, курсовая состоит из трех глав: теоретической; расчетной; итоговой.

Курсовая, имеющая общетеоретический характер и посвященная исследованию теоретических вопросов, состоит из двух глав: теоретической; итоговой

Курсовая, имеющая инженерно-технический характер состоит из двух разделов: раздел анализа и исследований и раздел реализации проекта.

Все главы(разделы) должны быть логически связаны друг с другом.

Также после каждой главы(раздела) подводятся итоги в виде краткого обобщения или вывода.

В разделе анализа и исследований курсовой работы автором обобщаются имеющиеся в открытом доступе сведения об объекте исследования. Изучается предметная область.

Исследуются подобные системы.

Оцениваются их преимущества и недостатки. В конце данного раздела производится постановка задачи

Эта часть посвящена описанию проблематики курсовой работы, ее практической значимости.

Здесь вкратце обосновывается актуальность выбранной темы.

Раздел реализации проекта курсовой работы является описанием исполнения проекта.

Здесь описываются алгоритмы, математические модели, применяемые и разрабатываемые методики.

В данном разделе описывается предполагаемый результат. предлагаются собственные идеи и разработки.

На этапе курсовой работы студент должен научиться грамотно излагать свои мысли, формулировать правильные выводы и обобщения. Вклад на этом этапе научно-исследовательской работы минимален. Здесь важно конкретизировать свои мысли, логично и последовательно представить свою точку зрения.

Заключение

Завершает текст курсовой работы заключение.

Объем заключения примерно равен или чуть больше объема введения.

В этой части автором излагаются полученные результаты.

Если в основном тексте это были обобщения, то здесь – четкие и лаконичные выводы.

В заключении указывается практическая значимость полученных результатов, дальнейшие перспективы в исследовании темы, формулируются рекомендации по применению полученных результатов исследования.

Список использованной литературы

После заключения в любой курсовой работе независимо от учебного заведения располагается список использованной литературы, в котором отражаются все библиографические источники, использованные автором при написании работы.

Список литературы оформляется в соответствии с ГОСТ 7.1-2003.

Название этого структурного элемента может различаться в зависимости от требований учебного заведения, но в целом допускается и «список литературы», и «список использованной литературы».

В списке литературы необходимо указать только те источники, которые использовались при работе над курсовой.

Обязательно должны быть расставлены ссылки в тексте работы.

Всего рекомендуется указывать около 10-15 литературных источников.

Приложения

Приложения – это необязательный структурный элемент курсовой работы.

Сюда выносятся материалы, уточняющие или подтверждающие отдельные элементы курсовой работы.

В отдельный структурный элемент приложения выделяют в том случае, если их количество больше двух.

В содержании приложения отображаются так: «Приложения» и номер страницы. Количество приложений в данном случае не имеет значения.

В приложения включаются: копии документации; копии планов, программ, чертежей; сопутствующий программный код; фотографии, крупные расчетные таблицы и другие материалы.

Объем приложений может составлять 30-40% от общего количества страниц в курсовой работе.

Однако возможно увеличение объема в случае такой необходимости. Каждому приложению присваивается номер.

Ссылки на все приложения должны присутствовать в основном тексте курсовой работы при каждом упоминании.

Иерархия расположения источников в списке следующая: нормативно-правовые акты располагаются по своей юридической значимости.

Сначала Конституция, затем Кодексы, затем Федеральные законы, затем Постановления Правительства и, наконец, местные законы.

Равные по силе документы располагаются по дате их публикации; учебные пособия, монографии, книги располагаются в алфавитном порядке; статьи тоже располагаются в алфавитном порядке;

В настоящее время с развитием интернета можно указывать в качестве литературных источников электронные ресурсы и электронные книги.

Но нужно учитывать, что ссылаться на обычную статью неизвестного автора в интернете нежелательно.

Используйте только проверенные и авторитетные источники.

Сетевая подсистема

Лекция 15 +

Сетевая подсистема Linux организует сетевой обмен пользовательских приложений и, как следствие, сетевое взаимодействие самих пользователей. Часть сетевой подсистемы, выполняющаяся в режиме ядра, естественным образом ответственна за управление сетевыми устройствами ввода-вывода, но, кроме этого, на нее также возложены задачи маршрутизации и транспортировки пересылаемых данных, которые решаются при помощи соответствующих сетевых протоколов.

Таким образом, именно ядерная часть сетевой подсистемы обеспечивает процессы средствами сетевого межпроцессного взаимодействия (network IPC).

Внеядерная часть сетевой подсистемы отвечает за реализацию сетевых служб, предоставляемых пользователям прикладные сетевые услуги, такие как передача файлов, обмен почтовыми сообщениями, удаленный доступ и т. д.

Сетевые интерфейсы, протоколы и сетевые сокеты

Непосредственное, физическое взаимодействие сетевых узлов через каналы связи между ними реализуется аппаратурой сетевых адаптеров.

Сетевые адAPTERы, как и любые другие устройства ввода-вывода, управляются соответствующими драйверами (листинг 6.1), реализуемыми в большинстве случаев в виде динамических модулей ядра.

Листинг 6.1. Драйвера сетевых устройств

```
lumpy@ubuntu:~$ lsblk
...
02:00.0 Network controller: Intel Corporation Centrino Advanced-N 6205 [Taylor Peak] (rev 34)
00:19.0 Ethernet controller: Intel Corporation 82579LM Gigabit Network Connection
(Lewisville) (rev 04)

lumpy@ubuntu:~$ lsblk -ks 02:00.0
02:00.0 Network controller: Intel Corporation Centrino Advanced-N 6205 [Taylor Peak] (rev 34)
Subsystem: Intel Corporation Centrino Advanced-N 6205 AGN
```

Kernel driver in use: iwlwifi

Kernel modules: iwlwifi ↵

lumpy@ubuntu:~\$ lspci -ks 02:00.0

00:19.0 Ethernet controller: Intel Corporation 82579LM Gigabit Network Connection (Lewisville) (rev 04)

Subsystem: Dell 82579LM Gigabit Network Connection (Lewisville)

Kernel driver in use: e1000e

Kernel modules: e1000e ↵

lumpy@ubuntu:~\$ modinfo iwlwifi e1000e | grep ^description

description: Intel(R) Wireless WiFi driver for Linux

description: Intel(R) PRO/1000 Network Driver

В отличие от несетевых устройств, большинство которых имеют специальный файл в каталоге /dev, сетевые устройства представляются в системе своими интерфейсами.

Список доступных интерфейсов, их параметры и статистику можно получить при помощи «классической» UNIX-команды ifconfig или специфичной для Linux команды ip (листинги 6.2 и 6.3).

Листинг 6.2. Сетевые интерфейсы (Unix ifconfig)

```
lumpy@ubuntu:~$ ifconfig -a

wlp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 192.168.0.101 netmask 255.255.255.0 broadcast 192.168.0.255
        inet6 fe80::5f2d:68c3:2c09:9a18 prefixlen 64 scopeid 0x20<link>
        ether 08:11:96:29:19:70 txqueuelen 1000 (Ethernet)
        ...
        ...

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
        inet 127.0.0.1 netmask 255.0.0.0
        inet6 ::1 prefixlen 128 scopeid 0x10<host>
        ...
        ...

eno1: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
        ether 5c:26:0a:85:a9:1a txqueuelen 1000 (Ethernet)
        ...
        ...
```

Листинг 6.3 Сетевые интерфейсы (Linux ip)

```
lumpy@ubuntu:~$ ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN ... qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eno1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc ... state DOWN ... qlen 1000
    link/ether 5c:26:0a:85:a9:1a brd ff:ff:ff:ff:ff:ff
3: wlp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP ... qlen 1000
    link/ether 08:11:96:29:19:70 brd ff:ff:ff:ff:ff:ff
```

```
lumpy@ubuntu:~$ ip addr show dev wlp2s0
3: wlp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 08:11:96:29:19:70 brd ff:ff:ff:ff:ff:ff
        inet 192.168.0.101/24 brd 192.168.0.255 scope global dynamic noprefixroute wlp2s0
            valid_lft 7121sec preferred_lft 7121sec
        inet6 fe80::5f2d:68c3:2c09:9a18/64 scope link noprefixroute
            valid_lft forever preferred_lft forever
```

За логическое взаимодействие (адресацию, маршрутизацию, обеспечение надежной доставки и пр.) отвечают сетевые протоколы, тоже в большинстве случаев реализуемые соответствующими модулями ядра.

Нужно отметить, что в примере из листинга 6.4 показан список динамически загруженных модулей, среди которых присутствует «нестандартный» TCP vegas, но нет IP, TCP, UDP и прочих «стандартных» протоколов стека TCP/IP.

На текущий момент времени сложно вообразить применение Linux без подключения к IP-сети, поэтому модули стандартных протоколов TCP/IP стека скомпонованы в ядро статически и являются частью «стартового» модуля.

Листинг 6.4. Драйвера сетевых протоколов

```
tumpy@ubuntu:~$ lsmod
```

Module	Size	Used by

iwlwifi	229376	0
mac80211	786432	1 iwlwifi
iwlwifi	290816	1 iwlwifi
cfg80211	622592	3 iwlwifi,mac80211

e1000e	249856	0
ptp	20480	1 e1000e


```
tumpy@ubuntu:~$ modinfo tcp_vegas bnepr mac80211 | grep ^description
```

description: PTP clocks support
description: Intel(R) Wireless WiFi Link AGN driver for Linux
description: IEEE 802.11 subsystem

Доступ процессов к услугам ядерной части сетевой подсистемы реализует интерфейс сетевых сокетов *Socket*, являющихся основным (и единственным) средством сетевого взаимодействия процессов в Linux.

Разные семейства (*address family*) сокетов соответствуют различным стекам сетевых протоколов. Например, стек TCP/IP v4 представлен семейством AF_INET, см. *ip*, стек TCP/IP v6 — семейством AF_INET6, см. *ip6*, и даже локальные (файловые) сокеты имеют собственное семейство — AF_LOCAL, см. *unix*.

Для просмотра статистики по использованию сетевых сокетов используют «классическую» UNIX-команду *netstat* или специфичную для Linux команду *ss*.

В листингах 6.5 и 6.6 иллюстрируется использование этих команд для вывода информации обо всех (-a, all) сокетах протоколов (-u, udp) UDP и (-t, tcp) TCP стека TCP/IP v4 (-4), порты которых выведены в числовом (-n, numeric) виде, а также изображены процессы (-p, process), их открывшие.

Листинг 6.5, Сетевые сокеты (UNIX netstat)

```
lumpy@ubuntu:~$ sudo netstat -4autp
```

Активные соединения с Интернетом (servers and established)

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State	PID/Program name
tcp	0	0	0.0.0.0:22	0.0.0.0:*	LISTEN	864/sshd
tcp	0	0	127.0.0.1:631	0.0.0.0:*	LISTEN	697/cupsd
tcp	0	0	127.0.0.1:5432	0.0.0.0:*	LISTEN	1039/postgres
tcp	0	0	0.0.0.0:25	0.0.0.0:*	LISTEN	1250/master
tcp	0	0	127.0.0.53:53	0.0.0.0:*	LISTEN	638/systemd-resolve
...	
tcp	0	0	192.168.0.101:22	192.168.0.103:57622	ESTABLISHED	6152/sshd: lumpy [pr
...	
udp	0	0	127.0.0.53:53	0.0.0.0:*		638/systemd-resolve
udp	0	0	192.168.0.101:68	0.0.0.0:*		684/NetworkManager

Сетевые сокеты идентифицируются парой адресов (собственным, local, и чужим, foreign. Это адрес удаленного приложения, с которым установлено соединение.), принятymi в их семействе.

Например, для семейства TCP/IP адрес сокета состоит из (сетевого) IP-адреса и (транспортного) номера порта, причем нули имеют специальное — «неопределенное» значение.

Прослушивающие сокеты используются «серверными» приложениями, пассивно ожидающими входящие соединения с ними.

Так, для прослушивающего (LISTEN) сокета 0 .0 .0 .0 в собственном IP-адресе означает, что он принимает соединения, направленные на любой адрес любого сетевого интерфейса, а 0.0.0.0 в чужом адресе указывает на то, что взаимодействие еще не установлено.

Для сокетов с установленным (ESTABLISHED) взаимодействием оба адреса имеют конкретные значения, определяющие участников взаимодействия, например 192.168.100.105:22 и 192.168.100.103: 57622.

Листинг 6.6. сетевые сокеты (Linux ss)

lumpy@ubuntu:~\$ sudo ss -4atupn						
Netif	State	Recv-Q	Send-Q	Local Address:Port	Peer Address:Port	
udp	UNCONN	0	0	127.0.0.53%lo:53	0.0.0.0:*	users:(("systemd-resolve",pid=638,fd=12))
udp	UNCONN	0	0	192.168.0.101%wlp2s0:68	0.0.0.0:*	users:(("NetworkManager",pid=684,fd=19))
...	
tcp	LISTEN	0	128	0.0.0.0:22	0.0.0.0:*	users:(("sshd",pid=864,fd=3))
tcp	LISTEN	0	5	0.0.0.1:631	0.0.0.0:*	users:(("cupsd",pid=697,fd=7))
tcp	LISTEN	0	128	127.0.0.1:5432	0.0.0.0:*	users:(("postgres",pid=1039,fd=3))
tcp	LISTEN	0	100	0.0.0.0:25	0.0.0.0:*	users:(("master",pid=1259,fd=13))
tcp	LISTEN	0	128	127.0.0.53%lo:53	0.0.0.0:*	users:(("systemd-resolve",pid=638,fd=13))
tcp	ESTAB	0	0	192.168.0.101:22	192.168.0.103:57622	users:(("sshd",pid=6234,fd=4),("sshd",pid=6152,fd=4))
...	

Из примеров листингов 6.5 и 6.6. видны 5 «слушающих» (LISTEN) сокетов TCP и 2 «несоединенных» (UNCONN) сокета UDP, открытых разными службами операционной системы.

Например, 22-й порт TCP открыл сервер sshd, PID = 864 службы удаленного доступа W:[SSH], а 5432-й порт TCP — сервер postgres, PID = 1039 службы реляционной СУБД W : [SQL].

Конфигурирование сетевых интерфейсов и протоколов

Ручное конфигурирование

Для функционирования разных стеков протоколов сетевым интерфейсам должны быть предварительно назначены корректные сетевые адреса и сконфигурированы прочие параметры, что может быть выполнено вручную администратором или автоматически специальными службами этих стеков.

Ручное назначение сетевых адресов стека TCP/IP выполняется при помощи команды `ifconfig` или `ip`, а простейшая диагностика — при помощи команды `ping`, как проиллюстрировано в листинге 6.7.

Листинг 6.7. Ручное конфигурирование сетевых интерфейсов

```
lumpy@ubuntu:~$ sudo ifconfig eno1 10.0.0.10 up
lumpy@ubuntu:~$ sudo ifconfig eno1
eno1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 10.0.0.10 netmask 255.0.0.0 broadcast 10.255.255.255
              inet6 fe80::66f6:6415:7455:6a0f prefixlen 64 scopeid 0x20<link>
                ether 08:00:27:c0:67:8f txqueuelen 1000 (Ethernet)
                  RX packets 0 bytes 0 (0.0 B)
                  RX errors 0 dropped 0 overruns 0 frame 0
                  TX packets 94 bytes 14932 (14.9 KB)
                  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
lumpy@ubuntu:~$ ping -c 1 10.0.0.10
PING 10.0.0.10 (10.0.0.10) 56(84) bytes of data.
64 bytes from 10.0.0.10: icmp_seq=1 ttl=64 time=0.032 ms

--- 10.0.0.10 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.032/0.032/0.032/0.000 ms
```

```
lumpy@ubuntu:~$ sudo ip address add 172.16.16.172/16 dev eno1
lumpy@ubuntu:~$ ip address show dev eno1
3: eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default
    qlen 1000
        link/ether 08:00:27:c0:67:8f brd ff:ff:ff:ff:ff:ff
        inet 10.0.0.10/8 brd 10.255.255.255 scope global eno1
            valid_lft forever preferred_lft forever
        inet 172.16.16.172/16 scope global eno1
            valid_lft forever preferred_lft forever

lumpy@ubuntu:~$ ping -c 1 172.16.16.172
PING 172.16.16.172 (172.16.16.172) 56(84) bytes of data.
64 bytes from 172.16.16.172: icmp_seq=1 ttl=64 time=1.27 ms

--- 172.16.16.172 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.270/1.270/1.270/0.000 ms
```

Просмотр таблиц маршрутизации и ручное конфигурирование IP-маршрутов выполняется посредством команды route или ip, а простейшая диагностика — при помощи команды traceroute.

В примере из листинга 6.8 показана процедура ручного добавления маршрута «по умолчанию» через интернет-шлюз 10.0.0.1 с последующей диагностикой доступности узлов за ним .

Листинг 6.8. Ручное конфигурирование таблицы маршрутизации

```
lumpy@ubuntu:~$ sudo ip route add 0.0.0.0/0 ① via 10.0.0.1
```

```
lumpy@ubuntu:~$ route -n
```

Таблица маршрутизации ядра протокола IP

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
0.0.0.0 ①	10.0.0.1 ②	0.0.0.0	UG	0	0	0	eno1
10.0.0.0	0.0.0.0	255.0.0.0	U	0	0	0	eno1
169.254.0.0	0.0.0.0	255.255.0.0	U	1000	0	0	eno1
172.16.0.0	0.0.0.0	255.255.0.0	U	0	0	0	eno1

```
lumpy@ubuntu:~$ ip route show
default ① via 10.0.0.1 ② dev eno1 proto static
10.0.0.0/8 dev eno1 proto kernel scope link src 10.0.0.1
172.16.0.0/16 dev eno1 proto kernel scope link src 172.16.0.1
169.254.0.0/16 dev eno1 scope link metric 1000

lumpy@ubuntu:~$ traceroute -m 50 bad.horse
traceroute to bad.horse (162.252.205.157), 30 hops max, 60 byte packets
 1  10.0.0.1 (10.0.0.1)  1.025 ms  1.080 ms  1.236 ms
                               ...
                               ...
 22  bad.horse (162.252.205.130)  191.988 ms  191.997 ms  178.848 ms
 23  bad.horse (162.252.205.131)  190.805 ms  195.160 ms  194.897 ms
 24  bad.horse (162.252.205.132)  199.199 ms  187.907 ms  201.063 ms
 25  bad.horse (162.252.205.133)  209.814 ms  203.633 ms  199.866 ms
 26  he.rides.across.the.nation (162.252.205.134)  209.300 ms  204.059 ms  211.089 ms
 27  the.thoroughbred.of.sin (162.252.205.135)  211.454 ms  208.207 ms  212.280 ms
 28  he.got.the.application (162.252.205.136)  208.660 ms  222.452 ms  210.522 ms
 29  that.you.just.sent.in (162.252.205.137)  215.037 ms  223.553 ms  223.043 ms
```

29 that.you.just.sent.in (162.252.205.137) 215.037 ms 223.553 ms 223.043 ms
30 it.needs.evaluation (162.252.205.138) 229.889 ms 231.036 ms 234.139 ms
31 so.let.the.games.begin (162.252.205.139) 226.369 ms 230.170 ms 223.952 ms
32 a.heinous.crime (162.252.205.140) 232.431 ms 231.814 ms 240.990 ms
33 a.show.of.force (162.252.205.141) 241.706 ms 233.070 ms 272.973 ms
34 a.murder.would.be.nice.of.course (162.252.205.142) 264.821 ms 239.704 ms 247.930 ms
35 bad.horse (162.252.205.143) 250.081 ms 244.279 ms 254.147 ms
36 bad.horse (162.252.205.144) 247.903 ms 258.675 ms 257.366 ms
37 bad.horse (162.252.205.145) 261.103 ms 253.861 ms 261.307 ms
38 he-s.bad (162.252.205.146) 267.135 ms 266.860 ms 258.031 ms
39 the.evil.league.of.evil (162.252.205.147) 262.974 ms 262.773 ms 275.656 ms
40 is.watching.so.beware (162.252.205.148) 278.567 ms 276.382 ms 277.577 ms
41 the.grade.that.you.receive (162.252.205.149) 276.158 ms 283.299 ms 284.268 ms
42 will.be.your.last.we.swear (162.252.205.150) 286.575 ms 286.470 ms 278.213 ms
43 so.make.the.bad.horse.gleeful (162.252.205.151) 283.040 ms 292.280 ms 290.042 ms
44 or.he.ll.make.you.his.mare (162.252.205.152) 300.872 ms 299.806 ms 289.688 ms
45 o_o (162.252.205.153) 294.548 ms 295.458 ms 295.030 ms

Автоматическое конфигурирование

За автоматическое конфигурирование сетевых интерфейсов отвечает менеджер сетевых подключений — системная служба `networkmanager`, отслеживающая физическую активацию сетевых адаптеров (подключение сетевого кабеля Ethernet или подключения к сети Wi-Fi) и взаимодействующая с другими службами, например со службой `wpa_supplicant` (для ассоциации с Wi-Fi точками доступа и аутентификации) или с локальной службой W:[DNS] `systemd-resolved`.

Кроме этого, менеджер сетевых подключений может запускать «подчиненные» службы, например W:[DHCP]-клиента `dhclient` или `dhpcd` для автоматического получения IP-адреса, простейший локальный DNS-сервер `dnsmasq` и т. д.

Для взаимодействия с менеджером сетевых подключений предназначены команда `nmcli` (см. также `nmcli-examples`), TUI-приложения `nmtui`, `nmtui-edit`, `nmtui-connect` и GUI-приложения `nm-applet`, `nm-connection-editor`, позволяющие опрашивать его состояние и управлять его действиями.

Листинг 6.9. Конфигурирование сетевых интерфейсов (автоматически)

```
lumpy@ubuntu:~$ nmcli dev
```

DEVICE	TYPE	STATE	CONNECTION
wlp2s0	wifi	подключено	474+
eno1	ethernet	отключено	--
lo	loopback	не настроено	--

```
lumpy@ubuntu:~$ nmcli dev show eno1
```

GENERAL.DEVICE:	eno1
GENERAL.TYPE:	ethernet
GENERAL.HWADDR:	08:00:27:C0:67:8F
GENERAL.MTU:	1500
GENERAL.STATE:	30 (отключено)
GENERAL.CONNECTION:	--
GENERAL.CON-PATH:	--
WIRED-PROPERTIES.CARRIER:	вкл.

```
lumpy@ubuntu:~$ nmcli conn
```

NAME	UUID	TYPE	DEVICE
464+	57cd20ce-098e-3b31-acd6-f50fd2e4db41	wifi	wlp2s0

```
lumpy@ubuntu:~$ sudo nmcli conn add type ethernet ifname eno1
```

Соединение «ethernet-eno1» (eadac6e2-eb71-43ee-9b34-86c2910a382c) добавлено.

```
lumpy@ubuntu:~$ nmcli conn
```

NAME	UUID	TYPE	DEVICE
464+	57cd20ce-098e-3b31-acd6-f50fd2e4db41	wifi	wlp2s0
ethernet-eno1	eadac6e2-eb71-43ee-9b34-86c2910a382c	ethernet	eno1

```
lumpy@ubuntu:~$ sudo nmcli conn up ethernet-eno1
```

Соединение успешно активировано (адрес действующего D-Bus:
/org/freedesktop/NetworkManager/ActiveConnection/6)

```
lumpy@ubuntu:~$ nmcli dev show eno1
```

GENERAL.DEVICE:	eno1
GENERAL.TYPE:	ethernet
GENERAL.HWADDR:	08:00:27:C0:67:8F
IP4.ADDRESS[1]:
IP4.GATEWAY:	10.0.2.1
IP4.ROUTE[1]:	dst = 0.0.0.0/0, nh = 10.0.2.1, mt = 101
IP4.ROUTE[2]:	dst = 10.0.2.0/24, nh = 0.0.0.0, mt = 10
IP4.DNS[1]:	10.0.2.1

```
lumpy@ubuntu:~$ ip a show dev eno1
3: eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP ... qlen 1000
link/ether 08:00:27:c0:67:8f brd ff:ff:ff:ff:ff:ff
inet 10.0.2.4/24 brd 10.0.2.255 scope global dynamic noprefixroute eno1
    valid_lft 954sec preferred_lft 954sec
inet6 fe80::9eca:df0a:5401:d34f/64 scope link noprefixroute
    valid_lft forever preferred_lft forever
```

```
lumpy@ubuntu:~$ ip route show
default via 10.0.2.1 brd enp0s8 proto dhcp metric 101
10.0.2.0/24 dev enp0s8 proto kernel scope link src 10.0.2.4 metric 101
169.254.0.0/16 dev enp0s3 scope link metric 1000
```

В листинге 6.9 проиллюстрирован результат работы менеджера подключений при подключении сетевого интерфейса eno1.

Полученные при помощи DHCP-клиента конфигурационные параметры были активированы автоматически: IP-адреса и маска назначены на интерфейс, а шлюз «по умолчанию» задан в соответствующем маршруте .

Служба имен и DNS/mDNS-резолверы

Основными идентификаторами сетевого взаимодействия в стеке протоколов TCP/IP являются числа — IP-адреса узлов и номера портов TCP/UDP, «человеческое» использование которых довольно неудобно. Если с 32-битным IPv4-адресом еще можно было совладать, то 128-битный адрес IPv6 не оставляет человеку практически никаких шансов

Использование строковых имен для узлов и портов приводит к необходимости отображать «человеческие» имена в «протокольные» числа и обратно, что возложено на службу имен (name service).

Как указывалось ранее, служба имен вообще предназначается для организации доступа приложений к свойствам каталогизируемых сущностей по их имени: к UID пользователя по имени его учетной записи, к IP-адресу по имени узла, к номеру порта TCP/UDP по имени сетевой службы, использующей его, и т. д.

Отображение имен сущностей на их свойства зачастую выполняется различными способами в разных каталогах, что определяется конфигурацией коммутатора службы имен `nsswitch.conf` (name service switch configuration) и наличием ее соответствующих модулей `libnss_*.so.?`.

В листинге 6.10 иллюстрируется такая конфигурация отображения имен узлов `hosts`, которая использует сначала файловую таблицу `/etc/hosts` — см. `hosts`, а затем — службы `W:[mDNS]` и `W:[DNS]`.

Аналогично, номера портов сетевых служб services отображаются сначала с использованием файла индексированной базы данных (соответствующий модуль libnss_db.so.2 оказался не установлен), а затем с использованием файловой таблицы /etc/services — см. services

Листинг 6.10 Служба имен и ее модули

```
lumpy@ubuntu:~$ grep hosts /etc/nsswitch.conf
①          ①↓ ②↓                      ③↓
hosts:      files mdns4_minimal [NOTFOUND=return] dns
lumpy@ubuntu:~$ grep services /etc/nsswitch.conf
services:   db files
lumpy@ubuntu:~$ find /lib/ -name 'libnss_*
...           ...           ...           ...
```

```
/lib/x86_64-linux-gnu/libnss_dns.so.2
/lib/x86_64-linux-gnu/libnss_files.so.2
...           ...           ...           ...
/lib/x86_64-linux-gnu/libnss_mdns4_minimal.so.2
...           ...           ...           ...
```

Файловые таблицы имен /etc/hosts и /etc/services имеют тривиальный формат ое, сопоставляющий имена узлов и сервисов — их IP-адресам и портам протоколов TCP и UDP, что проиллюстрировано в листинге 6.11.

Утилита службы имен getent, позволяющая выбирать указанную сущность по ее типу и имени, используется в качестве диагностики коммутатора службы имен и его модулей.

Листинг 6.11 Файловые таблицы имен

```
lumpy@ubuntu:~$ cat /etc/hosts
127.0.0.1      localhost
127.0.1.1      ubuntu

# The following lines are desirable for IPv6 capable hosts
::1      ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

```
lumpy@ubuntu:~$ grep http /etc/services
```

```
# Updated from https://www.iana.org/assignments/service-names-port-numbers/service-names-  
port-numbers.xhtml .
```

http	80/tcp	www	# WorldwideWeb HTTP
https	443/tcp		# http protocol over TLS/SSL
http-alt	8080/tcp	webcache	# WWW caching service
http-alt	8080/udp		

```
lumpy@ubuntu:~$ getent hosts ubuntu
```

127.0.1.1	ubuntu
-----------	--------

```
lumpy@ubuntu:~$ getent services 53/udp
```

domain	53/tcp
--------	--------

Соответствия IP-адресов именам «серверных» узлов, например публичных Web-, почтовых и прочих серверов, обычно регистрируются их администраторами в «таблицах» на ответственных (authoritative) серверах службы DNS.

Для доступа к ним соответствующий модуль службы имен (см., листинг 6.10) использует стандартный DNS-клиент (он же resolver, DNS-рэзолвер), в конфигурационном файле `resolv.conf` которого при статических настройках указываются IP-адреса ближайших кэширующих DNS-серверов, например серверов провайдера услуг Интернета.

Однако в примере из листинга 6.12 показано, что в качестве кэширующего DNS-сервера указан адрес 127.0.0.53 некоторой локальной службы DNS. Такой подход позволяет динамически управлять настройками (не меняя каждый раз файл `resolv.conf`) при реконфигурировании сетевых подключений, например при присоединении к другой Wi-Fi-сети.

В этом случае именно менеджер сетевых подключений сообщает новые DNS-параметры нового активированного подключения этой самой локальной службе DNS (которой на проверку оказывается `systemd-resolved`).

Для диагностики DNS-модуля службы имен (равно как и любого другого ее модуля) используется команда `getent`, а для непосредственной диагностики DNS-серверов — команда `host`.

Листинг 6.12. DNS-клиент

```
lumpy@ubuntu:~$ cat /etc/resolv.conf
# This file is managed by man:systemd-resolved(8). Do not edit.
...
# See man:systemd-resolved.service(8) for details about the supported modes of
# operation for /etc/resolv.conf.

nameserver 127.0.0.53
options edns0

lumpy@ubuntu:~$ sudo ss -4autpn sport = 53
Netid State Recv-Q Send-Q Local Address:Port  Peer Address:Port
udp    UNCONN 0        0      127.0.0.53%lo:53  0.0.0.0:*  users:(("systemd-resolve",pid=5932,...))
tcp    LISTEN 0       128    127.0.0.53%lo:53  0.0.0.0:*  users:(("systemd-resolve",pid=5932,...))

lumpy@ubuntu:~$ getent hosts bhv.ru
91.244.162.162 bhv.ru

lumpy@ubuntu:~$ host bhv.ru
bhv.ru has address 91.244.162.162
bhv.ru mail is handled by 50 relay2.peterlink.ru.
bhv.ru mail is handled by 30 relay1.peterlink.ru.

lumpy@ubuntu:~$ host 8.8.8.8
8.8.8.8.in-addr.arpa domain name pointer dns.google.
```

Сетевые устройства (принтеры, камеры, видеорегистраторы и пр.) и «клиентские» узлы локальных сетей, динамически получающие случайные IP-адреса при помощи DHCP, на ответственных серверах DNS почти никогда не регистрируются, а использование их имен в локальной сети (в «домене» .local) становится возможным благодаря службе W:[mDNS].

Серверы mDNS запускаются на каждом «клиентском» узле и регистрируют у себя соответствия собственных IP-адресов своему имени, а затем используют многоадресную (multicast) рассылку стандартных запросов DNS для получения информации друг у друга.

Сервером mDNS, как показано в примере из листинга 6.13, является avahi-daemon, реализующий еще и службу W:[DNS-SD] (DNS service discovery), которая позволяет узлам локальной сети обнаруживать (discover) услуги (service), предоставляемые другими узлами. При помощи avahi-browse проиллюстрирован список всех (-a, all) имен и типов услуг, объявленных узлами сети и сохраненных в локальном кэше (-c, cache), а также результаты (-g, resolve) запросов на получение информации об услугах.

Листинг 6.13. mDNS/DNS-SD-клиент

```
lumpy@ubuntu:~$ sudo ss -4autpn sport = :mdns
sudo ss -4autpn sport = :mdns
Netid State Recv-Q Send-Q Local Address:Port Peer Address:Port
udp UNCONN 0 0 0.0.0.0:5353 0.0.0.0:* users:(("avahi-daemon",pid=678,...))

lumpy@ubuntu:~$ avahi-browse -arcl
+ eth0 IPv4 HP LaserJet 700 M712 [4C6BF5] ↳ ⓘ ⓘ ↲ UNIX Printer local
... ...
+ eth0 IPv4 AXIS 211M - 00408C81D401 ... RTSP Realtime Streaming Server local
... ...
+ eth0 IPv4 NVR(SMB) ... Microsoft Windows Network local
+ eth0 IPv4 NVR(NFS) ... Network File System local
... ...
= eth0 IPv4 HP LaserJet 700 M712 [4C6BF5] ... UNIX Printer local
hostname = [NPI4C6BF5.local]
address = [192.168.17.68]
port = [515]
txt = ["Scan=F" "UUID=56ff35dd-1065-4e5e-9385-3c26b8946b79" "Color=F" "Duplex=T" "Binary=T"
"Transparent=T" "note=" "adminurl=http://NPI4C6BF5.local." "priority=40" "usb_MDL=HP LaserJet
700 M712" "usb_MFG=Hewlett-Packard" "product=(HP LaserJet 700 M712)" "ty=HP LaserJet 700 M712"
"URF=V1.1,CP99,RS600,MT1-2-3-5-12,W8,PQ4,IS20-21-22-23,DM1,081" "pd़l=application/postscript"
"rp=BINPS" "qtotal=4" "txtvers=1"]
... ...
= eth0 IPv4 AXIS 211M - 00408C81D401 ... RTSP Realtime Streaming Server local
hostname = [axis-00408c81d401.local]
address = [192.168.17.142]
```

```
port = [554]
txt = ["path=mpeg4/1/media.amp"]
= eth0 IPv4 NVR(SMB)
hostname = [NVR.local]
address = [192.168.17.90]
port = [445]
txt = []
= eth0 IPv4 NVR(NFS)
hostname = [NVR.local]
address = [192.168.17.90]
port = [2049]
txt = []
```

Microsoft Windows Network	local

Network File System	local

```
lumpy@ubuntu:~$ avahi-resolve --name NVR.local NPI4C6BF5.local axis-00408c81d401.local
NVR.local          192.168.17.90
NPI4C6BF5.local    192.168.17.68
axis-00408c81d401.local 192.168.17.142
```

```
lumpy@ubuntu:~$ getent hosts NVR.local NPI4C6BF5.local axis-00408c81d401.local
192.168.17.90      NVR.local
192.168.17.68      NPI4C6BF5.local
192.168.17.142     axis-00408c81d401.local
```

Для диагностики mDNS-модуля службы имен неизменно используется команда getent, а для непосредственной диагностики mDNS-сервера avahi-daemon — специальная команда avahi-resolve.

Сетевые службы

Служба SSH

Служба W:[SSH] предназначена для организации безопасного (secure) доступа к сеансу командного интерпретатора (shell) удаленных сетевых узлов.

Изначально разрабатывалась как замена небезопасным R-утилитам W:[Rlogin], W:[Rsh] и протоколу сетевого алфавитно-цифрового терминала W:[telnet].

При сетевом взаимодействии в «открытой» публичной сети, такой как Интернет, «безопасность» обычно понимают как конфиденциальность (плюс целостность) передаваемых данных и аутентичность (подлинность) взаимодействующих сторон.

Конфиденциальность данных, т. е. их недоступность некоторой третьей стороне, не участвующей во взаимодействии, обеспечивается в протоколе SSH при помощи симметричного шифрования с помощью общего сеансового ключа.

Симметричные алгоритмы шифрования используют для зашифровки и расшифровки информации один и тот же ключ, поэтому конфиденциальность целиком и полностью сводится к секретности ключа, т. е. его недоступности третьей стороне.

Сеансовый ключ устанавливается обеими взаимодействующими сторонами при помощи асимметричного алгоритма открытого согласования ключей (W: [протокол Диффи — Хеллмана]), использующего две пары дополнительных ключей.

Обе стороны взаимодействия случайным образом выбирают закрытые ключи и на их основе вычисляют парные открытые ключи, которыми публично обмениваются.

Общий (сеансовый) ключ вычисляется каждой стороной на основе своего закрытого ключа и чужого открытого ключа, что не может повторить третья сторона, т. к. может подслушать только передачу открытых ключей и не владеет закрытыми ключами участников взаимодействия.

При активном вмешательстве третьей стороны во взаимодействие путем перехвата и подмены сообщений согласования ключей злоумышленник может выдавать себя за другую сторону каждому из участников взаимодействия, став посредником, — см. W:[MITM] (*man in the middle*).

В этом случае взаимодействующие стороны согласуют свои сеансовые ключи со злоумышленником, предполагая, что согласовали их с подлинным участником взаимодействия.

Как следствие, все передаваемые данные «естественному» образом станут доступными злоумышленнику, причем наличие посредника никак не будет обнаружено.

Обеспечение подлинности (аутентичности) взаимодействующих сторон в протоколе SSH основывается на аутентификации сервера клиентом при помощи асимметричных алгоритмов цифровой подписи с использованием закрытого и открытого ключей сервера.

Закрытый ключ используется для подписывания сообщений, а парный ему открытый ключ — для проверки подписи, корректность которой удостоверяет в том, что сообщение было сгенерировано подлинным владельцем закрытого ключа.

В сообщение протокола Диффи — Хеллмана сервер добавляет свой открытый ключ цифровой подписи (так называемый *host key*), а само сообщение подписывает закрытым ключом.

Клиент извлекает этот открытый ключ из сообщения и с помощью проверки корректности его цифровой подписи удостоверяется в подлинности владельца вложенного ключа.

Остается только убедиться, что владельцем вложенного ключа и является целевой сервер.

В примере из листинга 6.14 иллюстрируется первое присоединение к SSH-серверу grex.org (162.202.67.158), в результате чего был получен открытый ключ алгоритма W: [ECDSA], который, возможно, действительно принадлежит этому серверу, а не злоумышленнику посередине соединения.

Единственный способ это проверить — заранее знать действительный открытый ключ SSH-сервера grex.org и побитно сверить его с присланным ключом, что довольно затруднительно сделать человеку.

Поэтому на практике сверяют короткие хэш-суммы действительного и присланного ключей, называемые «отпечатками пальца» (fingerprint), совпадение которых гарантирует совпадение¹ ключей.

Хэш-суммы открытых ключей SSH-серверов заранее известны и открыто публикуются, например для grex.org — на Web-странице <http://grex.cyberspace.org/faq.xhtml#sshfinger>.

После ручной сверки ключа при первом подключении он сохраняется в файле `~/.ssh/known_hosts` для автоматической сверки при последующих подключениях.

При этом подобрать другой ключ с такой же хэш-суммой практически невозможно.

Листинг 6.14. Первое присоединение к SSH-серверу

```
lumpy@ubuntu:~$ ssh jake@grex.org
The authenticity of host 'grex.org (75.61.90.157)' can't be established.
ECDSA key fingerprint is SHA256:pM03fe6UTyqtqzUMq5SmTrH5tqUuN9Wdv1wdpcEJhSU.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes ← ⏎
Warning: Permanently added 'grex.org,75.61.90.157' (ECDSA) to the list of known hosts.
jake@grex.org's password: P@ssw0rd ← ⏎
...
...
...
grex$ uname -a ←
OpenBSD grex.org 6.3 GENERIC#9 i386
grex$ whoami ←
jake
grex$ w ←
8:44PM up 41 days, 3:15, 8 users, load averages: 1.44, 1.46, 1.60
USER     TTY FROM          LOGGED IN   IDLE WHAT
cross    p0 166.84.136.80  01Nov19 2days -bash
mcd      p1 37.59.109.123  12Nov19 9days -zsh
fernand  pa 24.78.62.91   11Nov19 11days -ksh
cross    pi 166.84.136.80  Tue04PM 2days -bash
```

```
mattias  pq 81.233.210.67      18Oct19      0 /suid/bin/party
pbbl    ps 24.59.50.208       7:17PM      41 alpine
jake ❸●p8 93.100.207.82→❹ 8:44PM      0 w
lerxst  pg 47.50.84.206      03Nov1920days screen -RDD
grex$ tty
/dev/ttyp8 ❻
grex$ logout ↵
Connection to grex.org closed.
lumpy@ubuntu:~$
```

После успешного установления соединения SSH пользовательский сеанс аутентифицируется одним из способов, например с помощью пароля , а затем в интерактивном режиме запускается начальный командный интерпретатор аутентифицированного пользователя.

При этом на стороне сервера используется псевдотерминал и эмулируется терминальный вход в систему, считающийся сетевым.

Листинг 6.15. Выполнение отдельной команды

```
lumpy@ubuntu:~$ ssh jake@grex.org uptime  
jake@grex.org's password: Pa$$W0rd ↵  
8:47PM up 41 days, 3:17, 7 users, load averages: 2.34, 1.65, 1.64  
lumpy@ubuntu:~$
```

Кроме интерактивного сетевого входа, SSH позволяет удаленно выполнять отдельные команды (см. листинг 6.15), при этом псевдотерминал не используется, а терминальный вход не эмулируется.

Вместо этого стандартные потоки ввода-вывода STDIN, STDOUT и STDERR выполняемой команды просто перенаправляются через сетевое соединение ssh-клиенту.

Это зачастую используется для удаленного копирования файлов.

Например, в листинге 6.16 при помощи архиватора tar создается сжатый архив каталога /usr/src/sys на удаленном узле grex.org, а результат перенаправляется в локальный файл openbsd-kernel-source.tgz.

Листинг 6.16. Копирование при помощи ssh

```
lumpy@ubuntu:~$ ssh jake@grex.org tar czf - /usr/src/sys > openbsd-kernel-source.tgz
jake@grex.org's password: P@ssW0rd ↵
tar: Removing leading / from absolute path names in the archive
① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫
```

Аутентификация пользовательского сеанса по паролю имеет некоторое неудобство — необходимость вводить пароль заново при каждом новом соединении SSH.

Кроме того, пароль передается внутри соединения SSH, поэтому существует угроза подбора пароля злоумышленником.

Более прогрессивный способ аутентификации пользователя основан на использовании асимметричных алгоритмов цифровой подписи (аналогично аутентификации сервера в протоколе Диффи – Хелмана) и ключей пользователя.

Для этого открытый ключ пользователя единожды регистрируется на сервере, а при аутентификации подписывается его закрытым ключом и высыпается серверу повторно.

Сервер в свою очередь проверяет цифровую подпись присланного ключа и удостоверяется в подлинности его владельца, а побитное сравнение с заранее зарегистрированным ключом пользователя указывает на то, что владелец присланного ключа и есть целевой пользователь.

В листинге 6.17 иллюстрируется процедура применения ключей аутентификации пользователя.

Сначала при помощи команды ssh-keygen генерируется закрытый (private) W:[RSA]- ключ в локальном файле `~/.ssh/id_rsa` и парный ему открытый (public) ключ в локальном файле `~/.ssh/id_rsa.pub`.

Закрытый ключ защищается (шифруется) от несанкционированного использования парольной фразой (pass-phrase), которая в отличие от пароля никогда по сети не передается, а лишь обеспечивает доступ к самому ключу.

Нужно заметить, что использование пустой парольной фразы (как в примере) потенциально небезопасно, т. к. в случае кражи или разглашения ключей ими может воспользоваться любая третья сторона.

Затем при помощи команды ssh-copy-id(1) открытый ключ регистрируются на удаленном сервере `grep.org` в файле `~/.ssh/authorized_keys`, что происходит с предъявлением пароля. Последующие SSH-соединения аутентифицируются парой ключей, в результате отпадает необходимость вводить пароль.

Появляется необходимость вводить парольную фразу, но в примере она (непозволительно) пустая.

Листинг 6.17 Доступ по ключу

```
lumpy@ubuntu:~$ ssh-keygen
Generating public/private rsa key pair.

Enter file in which to save the key (/home/lumpy/.ssh/id_rsa): ↵
Enter passphrase (empty for no passphrase): ↵ ①
Enter same passphrase again: ↵ ②
Your identification has been saved in /home/lumpy/.ssh/id_rsa.
Your public key has been saved in /home/lumpy/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:zt1K3kUagoDuY4sUxqFMSZFngVxQQhP0Q4yBtDMov6w lumpy@ubuntu
...   ...   ...   ...
```

```
lumpy@ubuntu:~$ ssh-copy-id jake@grex.org
```

```
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/lumpy/.ssh/id_rsa.pub"
```

```
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that  
are already installed
```

```
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is  
to install the new keys
```

```
jake@grex.org's password: Password ↵
```

Number of key(s) added: 1

Now try logging into the machine, with: "ssh 'jake@grex.org'"

and check to make sure that only the key(s) you wanted were added.

```
lumpy@ubuntu:~$ ssh jake@grex.org ls
```

```
OPENME.txz
```

```
README.gz
```

```
lumpy@ubuntu:~$ ssh jake@grex.org zcat README.gz
```

А в файле OPENME.txz находится кое-что полезное ;)

```
lumpy@ubuntu:~$ ssh jake@grex.org file OPENME.txz
```

```
OPENME.txz: XZ compressed data
```

Использовать зашифрованные парольными фразами ключи и одновременно не вводить парольную фразу при каждом подключении позволяет SSH-агент ssh-agent, который удерживает в оперативной памяти расшифрованные единожды закрытые ключи пользователя и генерирует с их помощью цифровые подписи по запросу.

Листинг 6.18 иллюстрирует аутентификацию пользователя по ключу, защищенному парольной фразой, а затем передачу этого расшифрованного ключа агенту при помощи команды ssh-add, что позволяет избавиться от необходимости ввода парольной фразы ключа все время, пока запущен процесс ssh-agent (обычно – до окончания сеанса).

Наличие SSH-агента, запущенного в сеансе пользователя, обнаруживают две переменные окружения – SSH_AGENT_PID и SSH_AUTH_SOCK, содержащие соответственно PID агента и имя локального сокета для связи с ним.

Листинг 6.17 SSH-агент

```
lumpy@ubuntu:~$ ssh jake@grex.org file /usr/bin/ssh
Enter passphrase for key '/home/lumpy/.ssh/id_rsa': ...
/usr/bin/ssh: ELF 32-bit LSB shared object, Intel 80386, version 1

lumpy@ubuntu:~$ ssh-add @
Enter passphrase for /home/lumpy/.ssh/id_rsa: ...
Identity added: /home/lumpy/.ssh/id_rsa (/home/lumpy/.ssh/id_rsa)
lumpy@ubuntu:~$ ssh jake@grex.org ldd /usr/bin/ssh
/usr/bin/ssh:
      Start      End        Type  Open  Ref  GrpRef  Name
    17645000 37681000  exe   1     0    0      /usr/bin/ssh
    0b348000 2b3c5000  rlib   0     1    0      /usr/lib/libcrypto.so.43.1
    06757000 2675c000  rlib   0     1    0      /usr/lib/libutil.so.13.0
    00799000 207a1000  rlib   0     1    0      /usr/lib/libbz.so.5.0
    0f668000 2f698000  rlib   0     1    0      /usr/lib/libc.so.92.3
    0ab98000 0ab98000  ld.so  0     1    0      /usr/libexec/ld.so

lumpy@ubuntu:~$ env | grep SSH
SSH_AGENT_PID=21655
SSH_AUTH_SOCK=/tmp/ssh-Eehhsbx21654/agent.21654

lumpy@ubuntu:~$ ls -l /tmp/ssh-Eehhsbx21654/agent.21654
srw----- 1 lumpy lumpy 0 марта 28 23:07 /tmp/ssh-Eehhsbx21654/agent.21654
```

Протокол SSH получил широкое распространение далеко за рамками своего изначального предназначения.

Кроме непосредственного удаленного доступа, он используется другими командами для своих нужд. Например (см. листинг 6.19), команды scp и sftp используют ssh для безопасного сетевого копирования файлов, а команда rsync — для сетевой синхронизации (копирование изменившихся) файлов.

Все эти (и другие, подобные им) команды используют ssh для запуска некоторой «серверной» программы на удаленном узле (в частности, scp и rsync запускают сами себя, а sftp запускает sftp-server), с которой и взаимодействуют через защищенное соединение.

Лагтинг 6.19. Копирование файлов поверх SSH

```
lumpy@ubuntu:~$ scp *.pdf jake@grex.org:  
tcpdump.pdf                                         100%   37KB  37.3KB/s  00:00  
Wireshark_Display_Filters.pdf                      100%   38KB  38.0KB/s  00:00  
lumpy@ubuntu:~$ sftp jake@grex.org  
Connected to grex.org.  
sftp> ls  
OPENME.txz                                         README.gz  
Wireshark_Display_Filters.pdf                      linuxperf-tools.png  
tcpdump.pdf  
sftp> exit  
lumpy@ubuntu:~$ rsync -avrz jake@grex.org:/usr/share/man/man1 .  
receiving incremental file list  
man1/  
man1/acme-client.1  
man1/addr2line.1  
...  
man1/i386/fdformat.1  
man1/sparc64/  
man1/sparc64/fdformat.1  
man1/sparc64/mksuncd.1  
  
sent 9,529 bytes received 4,180,838 bytes 239,449.54 bytes/sec  
total size is 12,970,760 speedup is 3.10
```

Как оказывается на практике, использование «файловой» надстройки sftp-server для безопасного доступа к дереву каталогов удаленных узлов имеет достаточно широкое распространение.

В частности, терминальный файловый менеджер mc, W:[MidnightCommander], тоже является SSH:sftp-клиентом, что иллюстрирует листинг 6.20.

Более того, при помощи FUSE-файловых систем sshfs(см. листинг 3.29) или gvfs файлы SSH:sftp-серверов могут быть смонтированы в дерево каталогов так, что вообще любые программы смогут ими воспользоваться.

Листинг 6.20. Файловый менеджер с клиентом SSH (sftp)

Левая панель	Файл	Команда	Настройки	Правая панель
Список файлов		.[[^]]>	< /sh://jake@grex.org	.[[^]]>
Быстрый просмотр	C-x q	правки	'и Имя	Размер
Информация	C-x i	8 12:56	/..	-ВВЕРХ-
Дерево		15 2014	/.qt	марта 28 21:50
		3 12:27	/a	512 янв. 6 2012
		1 18:24	/afs	512 янв. 21 2012
Формат списка...		11 2013	/altroot	512 марта 6 2010
Порядок сортировки...		11 2013	~bbs	512 авг. 17 2011
Фильтр...		31 21:04	/bin	20 июня 8 2015
Выбор кодировки...	M-e	3 2015	/c	1024 янв. 22 2012
FTP-соединение...		11 2013	/cyberspace	512 янв. 21 2012
Shell-соединение...		9 18:37	/dev	39936 марта 31 19:07
Панелизация		29 01:03	/etc	4096 апр. 3 03:04
		1 17:24	/home	512 авг. 17 2011
		11 2013	/mnt	512 янв. 21 2012
Пересмотреть	C-g	22 10:17	/root	512 дек. 2 23:20
		11 2013	/sbin	2048 янв. 22 2012
-ВВЕРХ-				
75G/454G (16%)				

Совет: Макросы % работают даже в командной строке.

lumpy@ubuntu:~\$

1 Помощь 2 Члены 3 Документы 4 Правки 5 Копия 6 Перенос 7 ВК-од 8 Удалить 9 Меню 10 Выход [^]

Почтовые службы SMTP, POP/IMAP

Электронная почта, пожалуй, является самым ранним приложением сетевой подсистемы операционных систем семейства UNIX.

Изначально электронные письма пересыпались непосредственно между конечными сетевыми узлами при помощи службы W:[sendmail] с использованием протокола W: [SMTP], а для отправки или чтения писем применялась утилита mail.

Вместо sendmail может быть использована абсолютно любая реализация агента пересылки почты (W:[mail transport agent], MTA), например W:[postfix] или W:[exim], но обычно его функцию делегируют почтовым серверам провайдера услуг Интернета или серверам публичных сервисов типа yandex.ru или gmail.com.

Листинг 6.21 иллюстрирует «классическую» схему электронной почты с «локальным» МТА, использующую команду mail для составления исходящих О и чтения входящих е писем и команду mailq для просмотра очередей обработки почты.

Листинг 6.21 Обработка почты локальной почтовой системой

```
lumpy@ubuntu:~$ mail -s Тест dketov@gmail.com
```

Cc: ↵

Это тест ;) ↵

^D

```
lumpy@ubuntu:~$ mailq
```

Mail queue is empty

```
lumpy@ubuntu:~$ mail
```

Mail version 8.1.2 01/15/2001. Type ? for help.

"/var/mail/lumpy" ↵: 1 message 1 new

```
>N 1 MAILER-DAEMON@ubu Thu Jan 7 15:09 77/2653 Undelivered Mail Returned to Sender  
& 1 ↵
```

Message 1:

From: MAILER-DAEMON Thu Jan 7 15:09:35 2016

X-Originat-To: lumpy@ubuntu

Date: Thu, 7 Jan 2016 15:09:35 +0300 (MSK)

From: MAILER-DAEMON@ubuntu (Mail Delivery System)

Subject: Undelivered Mail Returned to Sender

To: lumpy@ubuntu

...

...

...

...

This is the mail system at host ubuntu. ↵

I'm sorry to have to inform you that your message could not be delivered to one or more recipients. It's attached below.

...
<dketov@gmail.com>: host gmail-smtp-in.l.google.com[74.125.205.27] said:

550-5.7.1 [95.55.94.237] The IP you're using to send mail is not Authorized to 550-5.7.1 send email directly to our servers. Please use the ↵ - SMTP relay at your 550-5.7.1 service provider ↵ instead. Learn more at 550-5.7.1 <https://support.google.com/mail/answer/10336> tw4si41552991lbb.77 - gsmtp (in reply to end of DATA command)

...
& q ↵

Saved 1 message in /home/lumpy/nbox ↵

lumpy@ubuntu:~\$ mail handy@happytreefriends.ru

Cc: ↵

Subject: Cm. hands(4) ... ↵
... npo /dev/hands ;) ↵

^D

lumpy@ubuntu:~\$ mailq

-Queue ID- --Size-- ----Arrival Time---- -Sender/Recipient-----
8B85027304* 341 Sun Nov 24 00:38:27 lumpy@ubuntu

handy@happytreefriends.ru

Современные требования к условиям корректной пересылки почтовых сообщений (например в листинге 6.21) зачастую оказываются чрезвычайно строгими, а содержание собственной локальной почтовой системы — неоправданно сложным.

В большинстве случаев конечные пользователи пользуются услугами «внешних» почтовых серверов, организующих полный цикл обработки почты — от приема исходящих сообщений до обслуживания почтовых ящиков.

Исходящие сообщения отправляются таким серверам с помощью протокола W:[SMTP], а доступ к почтовым ящикам — при помощи протоколов W: [POP3] или W: [IMAP].

В примере из листинга 6.22 показан терминальный клиент современных почтовых систем `mutt`, поддерживающий «защищенные» протоколы электронной почты SMTPs, IMAPs и POPs, использующие протокол W: [SSL] для криптозащиты сетевых соединений, использующий весьма похожие на SSH способы обеспечения конфиденциальности данных и аутентичности взаимодействующих сторон.

Для отправки сообщений используются учетная запись пользователя lumpy.noose и сервер «исходящих» сообщений sntp.yandex.ru, принимающий почту по протоколу SMTPs , а для чтения писем из почтового ящика пользователя lumpy.noose могут быть использованы протоколы IMAPS или POPs и серверы «входящих» сообщений inap.yandex.ru и pop.yandex.ru, соответственно.

Листинг 6.22. Обработка почты публичной почтовой службой

❶ lumpy@ubuntu:~\$ mutt -s Тест dketov@gmail.com

...

lumpy@ubuntu:~\$ cat ~/.muttrc

Заполнить

set from=lumpy.moose@yandex.ru

set smtp_url=smt�://lumpy.moose@smtp.yandex.ru

① ↴

❷ lumpy@ubuntu:~\$ mutt -f imap://lumpy.moose@imap.yandex.ru

② ↴

↳ Определяется адрес сервера imap.yandex.ru...

↳ Устанавливается соединение с imap.yandex.ru...

↳ SSL/TLS-соединение с использованием TLS1.3 (ECDHE-RSA/AES-256-GCM/AEAD)

↳ Пароль для lumpy.moose@imap.yandex.ru: ↵

q:Выход d:Удалить u:Восстановить s:Сохранить m:Создать g:Ответить a:Всем

1 Ян 07 Яндекс (9,9K) Соберите всю почту в этот ящик

2 Ян 07 Команда Яндекс. (15K) Как читать почту с мобильного

3 Н + Мар 30 Яндекс.Паспорт (8,4K) Доступ к аккаунту восстановлен

--Mutt: imap://lumpy.moose@imap.yandex.ru/INBOX [Msgs:3 New:1 33K]---(threads/date)-(all)---

*

❸ lumpy@ubuntu:~\$ mutt -f pop3://lumpy.moose@pop.yandex.ru

③ ↴

...

Служба WWW

Служба W:[WWW] знакома каждому современному пользователю и в комментариях особо не нуждается. Одной ее заметной особенностью в Linux, пожалуй, является существование терминальных Web-браузеров links, lynx, elinks и w3m, позволяющих работать с «текстовой» частью гипертекстовых Web-ресурсов, что проиллюстрировано с помощью lynx в примере из листинга 6.23

Листинг 6.23. Терминальные браузеры lynx, links и w3m.

```
lumpy@ubuntu:~$ Lynx http://www.kernel.org
```

#The Linux Kernel Archives Atom Feed Latest Linux Kernel Releases

The Linux Kernel Archives (p1 of 4)

The Linux Kernel Archives

- * About
- * Contact us
- * FAQ
- * Releases
- * Signatures
- * Site news

Protocol	Location
HTTP	https://www.kernel.org/pub/
GIT	https://git.kernel.org/
RSYNC	rsync://rsync.kernel.org/pub/

Latest Stable Kernel:
Download 5.3.12

Кроме Web-браузеров, предназначенных для интерактивной работы пользователей, в сценариях на языке командного интерпретатора зачастую используются неинтерактивные пользовательские агенты wget и curl, позволяющие автоматизировать Web-взаимодействие.

Так, например, в листинге 6.24 при помощи wget показано скачивание файла в режиме «с докачкой» (-c, continue), а curl применяется для обращения к Google Geocoding API.

Листинг 6.24. Пользовательские агенты wget и curt

```
lumpy@ubuntu:~$ wget -c http://www.brendangregg.com/Perf/linuxperf-tools.png
--2019-11-24 08:54:08--  http://www.brendangregg.com/Perf/linuxperf-tools.png
Распознаётся www.brendangregg.com (www.brendangregg.com)... 184.168.188.1
Подключение к www.brendangregg.com (www.brendangregg.com)|184.168.188.1|:80... соединение установлено.
НПР-запрос отправлен. Ожидание ответа... 200 OK
Длина: 523561 (511K) [image/png]
Сохранение в: «linuxperf-tools.png»

42% [=====] 224 157 --.-K/s  за 11s
```

2019-11-24 00:54:19 (19,5 kB/s) - Ошибка чтения, позиция 224157/523561 (Время ожидания соединения истекло).
Продолжение попыток.

--2019-11-24 00:54:20-- (попытка: 2) <http://www.brendangregg.com/Perf/linuxperftools.png>

Подключение к www.brendangregg.com (www.brendangregg.com)|184.168.188.1|:80... соединение установлено.

НПТР-запрос отправлен. Ожидание ответа... 206 Partial Content

Длина: 523561 (511K), 299404 (292K) осталось [image/png]

Сохранение в: «linuxperftools.png»

100%[=====] 523 561 213K/s за 1,4s
↓

2019-11-24 00:54:22 (747 kB/s) - «linuxperftools.png» сохранён [523561/523561]

lmpy@ubuntu:~\$ curl -I http://man7.org/tlpi/download/TLPI-24-Process_Creation.pdf

HTTP/1.1 200 OK

Date: Sat, 23 Nov 2019 22:08:17 GMT

Server: Apache

Last-Modified: Thu, 21 Nov 2019 14:26:50 GMT

ETag: "171-31f5f5-597dc16857e80"

Accept-Ranges: bytes

Content-Length: 3274229

Connection: close

Content-Type: application/pdf

Служба FTP

Протокол W:[FTP] является «ископаемым» даже по сравнению с W:[SMTP], однако все еще широко используется для организации доступа к обширным файловым хранилищам.

Основная особенность протокола — отделение потока команд от потоков данных, что позволяет организовать параллельную передачу нескольких файлов одновременно.

За счет этой особенности появляется возможность (практически не используемая, как небезопасная) передачи файлов не между файловым сервером и клиентом (как «обычно»), а между двумя (!) файловыми серверами, см. W:[FXP].

В листинге 6.25 иллюстрируется lftp — один из самых распространенных терминальных FTP-клиентов, имеющий массу полезных возможностей, как то: «задачи» заднего фона, зеркалирование файловых поддеревьев (включая FXP), неинтерактивная работа и т. д.

Листинг 6.25. FTP-клиент lftp

```
lumpy@ubuntu $ lftp cdimage.debian.org  
lftp cdimage.debian.org:~> cd /cdimage/ports/latest/hurd-i386/current ↵  
cd ok, каталог=/cdimage/ports/latest/hurd-i386/current  
lftp cdimage.debian.org:/..../hurd-i386/current> ls *.iso ↵  
-rw-r--r-- 1 ftp ftp 670121984 Feb 20 2019 debian-sid-hurd-i386-CD-1.iso  
-rw-r--r-- 1 ftp ftp 1764358144 Feb 20 2019 debian-sid-hurd-i386-DVD-1.iso  
-rw-r--r-- 1 ftp ftp 174952448 Feb 20 2019 debian-sid-hurd-i386-NETINST-1.iso  
-rw-r--r-- 1 ftp ftp 30199808 Feb 20 2019 mini.iso  
lftp cdimage.debian.org:/..../hurd-i386/current> get debian-sid-hurd-i386-DVD-1.iso & ↵  
① ↴  
[0] get debian-sid-hurd-i386-DVD-1.iso &  
`debian-sid-hurd-i386-DVD-1.iso' в позиции 0  
lftp cdimage.debian.org:/..../hurd-i386/current> get mini.iso & ↵  
② ↴
```

```
[1] get mini.iso &
    'mini.iso' в позиции 0

lftp cdimage.debian.org:/.../hurd-i386/current> jobs ↵
[1] get mini.iso -- 907.5 Киб/с
    'mini.iso' в позиции 1573976 (5%) 907.5Кб/с овл:31с [Получение данных]
[0] get debian-sid-hurd-i386-DVD-1.iso -- 2.95 Миб/с
    'debian-sid-hurd-i386-DVD-1.iso' в позиции 18610948 (1%) 2.95Мб/с овл:9м [Получение
        данных]
```

```
lftp cdimage.debian.org:/.../hurd-i386/current> quit ↵
[12334] Переход в фоновый режим для завершения работы заданий...
```

```
lumpy@ubuntu $ lftp -c attach 12334
[12334] Присоединился к терминалу.

lftp cdimage.debian.org:/cdimage/ports/latest/hurd-i386/current> jobs ↵
[0] get debian-sid-hurd-i386-DVD-1.iso -- 498.6 Киб/с
    'debian-sid-hurd-i386-DVD-1.iso' в позиции 718802944 (40%) овл:6м [Получение данных]

lftp cdimage.debian.org:/cdimage/ports/latest/hurd-i386/current> quit ↵
[12334] Переход в фоновый режим для завершения работы заданий...
```

Кроме массы специализированных FTP-клиентов ftp, lftp, ncftp, gftp, протокол FTP поддерживается и другими программными средствами, скажем различными файлами.

6.26

пс.

Левая панель	Файл	Команда	Настройки	Правая панель
Список файлов			.[^]>	< /ftp://mirrorg.yandex.ru > —.[^]>
Быстрый просмотр	C-x q	2 05:02	правки	'и Имя Размер Время правки
Информация	C-x i	18 21:00	/..	-ВВЕРХ- марта 28 21:50
Дерево		30 14:08	/.ping	4096 марта 31 2014
		31 21:00	/altlinux	4096 апр. 2 05:02
		2 13:33	/altlinux-beta	4096 марта 18 21:00
Формат списка...		12 13:36	/altlinux-lightly	4096 марта 30 14:08
Порядок сортировки...		18 21:00	/altlinux-erkits	4096 марта 31 21:00
Фильтр...		17 2007	/archlinux	4096 апр. 2 13:33
Выбор кодировки...	M-e	15 20:51	/archlinux-arm	4096 окт. 12 13:36
FTP-соединение...		2 02:34	/archserver	4096 марта 18 21:00
Shell-соединение...		2 12:06	/asplinux-tigro	4096 сент. 17 2007
Панелизация		2 13:18	/astra	4096 марта 15 20:51
		13 09:26	/calculate	4096 апр. 2 02:34
Пересмотреть	C-Г	2 09:36	/centos	4096 апр. 2 12:06
		2 01:20	/debian	4096 апр. 2 13:18
			/debian--kports	4096 марта 13 09:26
			-ВВЕРХ-	
/archlinux				

Совет: Внешний просмотрщик можно выбрать с помощью переменной оболочки PAGER.
Ubuntu:~\$ [^]

1 Помощь 2 Меню 3 Проверка 4 Правка 5 Копия 6 Перенос 7 Назад 8 Удалить 9 Меню MS 10 Выход

Кроме всего прочего, клиентами FTP являются еще и внеядерные FUSE-файловые системы curlftpfs (см. листинг 3.29) или gvfs, позволяющие монтировать файлы FTP-серверов в дерево каталогов для их использования вообще любыми программами.

Служба NFS

Служба W: [Network File System] изначально разрабатывалась для прозрачного сетевого использования файловых систем сервера так, как будто они были непосредственно примонтированы в дерево каталогов клиента.

В отличие от FTP transfer-протокола, предназначенного для скачивания (transfer) файлов, протокол NFS является ретранслятором системных вызовов open, close, read, write, seek и прочих с клиента на сервер.

Это позволяет клиенту выполнять операции чтения/записи с любой частью файла, без его передачи целиком.

NFS-клиент

Клиент протокола NFS непосредственно реализован в ядре Linux при помощи модулей nfs, nfsv2/nfsv3/nfsv4 и используется с помощью штатной операции монтирования `mount`, тем самым делая доступными серверные файлы любым клиентским программам.

В примере из листинга 6.27 показана процедура монтирования файловой системы /Qmultimedia NFS-сервера NVR.local в каталог /mnt/network/nvr/Qmm при помощи протокола NFS v3 (-t nfs -o vers=3).

Для получения списка экспортируемых (-e, `export`) сервером файловых систем вызывается команда `showmount`, которая также является специализированным NFS-клиентом.

Листинг 6.27. Монтирование NFS

```
Lumpy@ubuntu:~$ showmount -e NVR.local
Export list for NVR.local:
/Qweb *
/Qusb *
/Qrecordings *
/Qmultimedia *
/Qdownload *
/Public *
/Network Recycle Bin 1 *

Lumpy@ubuntu:~$ sudo mkdir -p /mnt/network/nvr/Qmm
Lumpy@ubuntu:~$ sudo mount -t nfs -o vers=3 NVR.local:/Qmultimedia /mnt/network/nvr/Qmm
Lumpy@ubuntu:~$ findmnt -t nfs
TARGET                SOURCE          STYPE   OPTIONS
/mnt/network/nvr/Qmm  NVR.local:/Qmultimedia  nfs    rw,relatime,vers=3,...
```



```
Lumpy@ubuntu:~$ ls /mnt/network/nvr/Qmm
-rw-r--r-- 1 toothy  users  678696 авг.  20 2014 IMG_20140719_125651.jpg
-rw-r--r-- 1 toothy  users  649685 авг.  20 2014 IMG_20140719_125713.jpg
            ...           ...
-rw-r--r-- 1 toothy  users  814607 авг.  20 2014 IMG_20140820_111355.jpg

Lumpy@ubuntu:~$ id toothy
uid=1010(toothy) gid=1012(toothy) группа=1012(toothy)
```

Необходимо отметить, что права доступа и идентификаторы UID/GID владельцев файлов передаются NFS-протоколом в неизменном виде, поэтому базы пользовательских учетных записей всех клиентов (и сервера) должны быть согласованы, например, при помощи их централизованного хранения в каталоге LDAP.

NFS-сервер

Функционирование NFS-сервера имеет свою специфику, связанную с использованием NFS-протоколом принципа RPC (remote procedure call, удаленного вызова процедур) в реализации W:[SUNRPC].

Серверы, использующие SUN RPC, не имеют «закрепленного» номера порта TCP/UDP, а используют произвольный, случайно выбираемый порт.

Как, например, порт 22 закреплен за службой SSH, порт 25 — за SMTP, а порт 80 — за HTTP протоколом службы WWW.

Вместо этого в SUN RPC закрепляются номера «программ» (program), предоставляющих определенные «услуги» (service), а соответствующий порт регистрируется в служебной RPC-программе portmapper, что проиллюстрировано в листинге 6.28 при помощи утилиты `grcinfo`.

Листинг 6.28. Удаленный вызов процедур RPC: динамические номера портов и portmapper.

```
lupru@ubuntu:~$ rpcinfo -p NVR.local
```

program	vers	proto	port	service
100000	3	tcp	111	portmapper
100000	3	udp	111	portmapper
100003	3	tcp	2049	nfs
100003	3	udp	2049	nfs
100005	3	udp	53964	mountd
100005	3	tcp	39835	mountd

При помощи portmapper организуется обнаружение NFS-серверов в локальной сети посредством широковещательного (-b, broadcast) поиска зарегистрированных RPC-программ NFS v3 (листинг 6.29).

Порт самой RPC-программы portmapper все же закреплен за номером 111 TCP/UDP, что позволяет клиентам обращаться к portmapper сервера и находить порты других RPC-программ по их номерам.

Кроме этого, некоторые серверы NFS (например, сетевые устройства хранения данных W:[NAS] или сетевые видеорегистраторы W:[NVR]) регистрируются в службе mDNS/DNS-SD, что обнаруживается при помощи `avahi-browse`.

Листинг 6.29 Обнаружение NFS-сервера

```
lumpy@ubuntu:~$ rpcinfo -b nfs 3
192.168.17.90.8.1      NVR.local
...
lumpy@ubuntu:~$ avahi-browse -rcl _nfs._tcp
...
+  eth0 IPv4 NVR(NFS)
...
=  eth0 IPv4 NVR(NFS)
    hostname = [NVR.local]
    address = [192.168.17.90]
    port = [2049]
    txt = []
...
...
• Network File System
...
Network File System
...
local
local
```

Сервер NFS предоставляет клиентам некоторое количество RPC-услуг (-s, services), показанных в листинге 6.30, при помощи rpcinfo.

Выделяют базовые RPC-программы mountd и nfs , позволяющие монтировать файловые системы NFS и обращаться к их файлам, и дополнительные RPC-программы nlockmgr и status, реализующие механизм блокировки файлов.

Листинг 6.30 RPC-программы службы NFS

```
lumpy@ubuntu:~$ rpcinfo -s NVR.local
```

program	version(s)	netid(s)	service	owner
100000	4,3,2	tcp6,tcp,udp,udp6	portmapper	superuser
100003	4,3,2	udp6,tcp6,udp,tcp	① nfs	superuser
100005	3,2,1	tcp6,udp6,tcp,udp	② mountd	superuser
100021	4,3,2,1	tcp6,udp6,tcp,udp	③ nlockmgr	superuser
100024	1	tcp6,udp6,tcp,udp	④ status	superuser

Служба SMB/CIFS

Служба W:[CIFS] (common internet file system), заимствованная из семейства операционных систем Windows, предназначена (аналогично «родной» NFS) для совместного использования файлов.

Основным протоколом службы CIFS является протокол SMB (server message blocks), который аналогично NFS ретранслирует системные вызовы к файлам.

Имена NetBIOS

Отличительной особенностью протокола SMB, доставшейся ему в наследство от транспорта W: [NetBIOS], является возможность использования еще одного вида имен узлов (в дополнение к DNS- и mDNS-именам) — так называемых «имен NetBIOS» — и собственной службы NBNS (netbios name service), отображающей имена NetBIOS на IP-адреса. Служба NBNS похожа на DNS и mDNS одновременно, но несовместима с ними.

Листинг 6.31 Имена узлов NetBios и их IP-адреса

```
Lumpy@ubuntu:~$ nmblookup WINXP
querying WINXP on 192.168.100.255
192.168.100.3 WINXP<00>
```

В листинге 6.31 иллюстрируется использование утилиты `nmblookup(l)`, предназначеннай для диагностики службы NBNS, при помощи которой «плоское» имя NetBIOS HINXP отображается на соответствующий ему IP-адрес. Именно при помощи службы NBNS и широковещательного поиска специальных «групповых» имен NetBIOS реализуется основной способ обнаружения CIFS-серверов локальной сети, что выполняет утилита `smbtree`, иллюстрируемая в листинге 6.32.

В редких отдельных случаях серверы CIFS обнаруживаются зарегистрированными в службе mDNS/DNS-SD, что характерно (как и в случае серверов NFS) для сетевых устройств хранения данных W:[NAS] или сетевых видеорегистраторов W:[NVR].

Листинг 6.31 Обнаружение CIFS-серверов

```
lumpy@ubuntu:~$ smbtree -N
```

WORKGROUP

\WINXP

\WINXP\CS	Стандартный общий ресурс
\WINXP\ADMIN\$	Удаленный Admin
\WINXP\media	Фото, видео, и т. д.
\WINXP\DS	Стандартный общий ресурс
\WINXP\IPC\$	Удаленный IPC

\NVR

\NVR\Qrecordings
\NVR\Qmultimedia
\NVR\Qdownload
\NVR\IPC\$

```
lumpy@ubuntu:~$ avahi-browse -rcl _smb._tcp
```


+	eth0 IPv4 NVR(SMB)	...	Microsoft Windows Network	...	local
=	eth0 IPv4 NVR(SMB)	...	Microsoft Windows Network	...	local
	hostname = [NVR.local]				
	address = [192.168.17.90]				
	port = [445]				
	txt = []				

CIFS-клиенты

Различают две разные реализации клиента CIFS — внеядерную `smbclient` (аналогичную «интерактивным» FTP-клиентам) и ядерную (аналогичную NFS-клиенту), реализуемую модулем ядра `clfs`.

Использование ядерного модуля позволяет монтировать общие файловые ресурсы (`share`) серверов CIFS непосредственно в дерево каталогов клиента, что дает возможность любым его программам использовать серверные файлы.

В примерах из листинга 6.33 показано использование CIFS-клиента `smbclient` для получения списка (-L, `list`) разделяемых ресурсов (`share`) узла WINXP, равно как и для подключения к его публичному (-N, `no password`) разделяемому ресурсу `media` с последующим скачиванием файлов целиком.

Листинг 6.33. Клиент SMB/CIFS

```
Lumpy@ubuntu:~$ smbclient -NL //WINXP
Domain=[WINXP] OS=[Windows 5.1] Server=[Windows 2000 LAN Manager]

      Sharename      Type      Comment
      -----
      C$            Disk      Стандартный общий ресурс
      D$            Disk      Стандартный общий ресурс
      ADMIN$        Disk      Удаленный Admin
      IPC$          IPC       IPC Service
      media         Disk      Фото, видео, и т. д.

Lumpy@ubuntu:~$ smbclient -N //WINXP/media
Domain=[WINXP] OS=[Windows 5.1] Server=[Windows 2000 LAN Manager]
smb: \> cd DCIM\ <
smb: \DCIM\> dir <
.
..
Dd595.jpg
Dd596.jpg
Dd680.jpg
Dd681.jpg

              D          @    Thu Jan  7 20:57:36 2016
              D          @    Thu Jan  7 20:57:36 2016
A 4494092   Sun Feb 13 16:24:04 2011
A 3842680   Sun Feb 13 16:14:56 2011
...          ...          ...
A 4087313   Sun Feb 13 03:10:45 2011
A 4108278   Sun Feb 13 15:58:38 2011

      61192 blocks of size 1048576. 10915 blocks available

smb: \DCIM\> get Dd680.jpg <
getting file \DCIM\Dd680.jpg of size 4087313 as Dd680.jpg (72572,9 KiloBytes/sec)
(average 72573,0 KiloBytes/sec)
smb: \DCIM\> quit <
```

Листинг 6.34 иллюстрирует использование ядерного модуля cifs для монтирования публичного (-o guest) ресурса media с узла WINXP в каталог /mnt/network/winxp/media.

Клиент smbclientfl) имеет встроенный механизм NBNS, поэтому без проблем подключается к узлу MINXP, а при монтировании cifs механизм NBNS недоступен, что требует подсказки в виде IP-адреса сервера.

Листинг 6.34 Монтирование ресурса SMB/CIFS

```
lumpy@ubuntu:~$ sudo mkdir -p /mnt/network/winxp/media
lumpy@ubuntu:~$ sudo mount -t cifs -o guest //WINXP/media /mnt/network/winxp/media
mount error: could not resolve address for WINXP: Unknown error
lumpy@ubuntu:~$ nmblookup WINXP
querying WINXP on 192.168.100.255
192.168.100.3 WINXP<00>
lumpy@ubuntu:~$ sudo mount -t cifs -o guest,ip=192.168.100.3 //WINXP/media /mnt/network/winxp/media
lumpy@ubuntu:~$ findmnt -t cifs
TARGET                SOURCE          FSTYPE OPTIONS
/mnt/network/winxp/media  //WINXP/media cifs   rw,relatime,vers=1.0,cache=strict,...
lumpy@ubuntu:~$ ls -l /mnt/network/winxp/media/DCIM/
итого 346228
-rwxr-xr-x 1 root root 4494092 февр. 13 2011 Dd595.jpg
... ...
-rwxr-xr-x 1 root root 4108278 февр. 13 2011 Dd681.jpg
```

Средства сетевой диагностики

Диагностика сетевого обмена существенно облегчает решение разнообразных задач, связанных с эксплуатацией или разработкой сетевых приложений.

К сетевым диагностическим специальным средствам относят анализаторы пакетов и сетевые сканеры, которые применяются самостоятельно или вместе с трассировщиками системных и библиотечных вызовов.

Анализаторы пакетов tcpdump и tshark

Анализаторы пакетов предназначены для перехвата данных, поступающих из сети на сетевые интерфейсы или отправляющиеся в сеть с сетевых интерфейсов.

Современные анализаторы пакетов, кроме собственно захвата пакетов, осуществляют их детальный протокольный разбор, а также позволяют отфильтровывать подлежащие анализу пакеты по ряду критериев.

В листинге 6.35 показан пример использования наиболее распространенного, «классического» анализатора пакетов `tcpdump`, анализирующего пакеты на интерфейсе (-г, `interface`) `wlp2s0`, адресованные порту `port 53`.

Листинг 6.35. Анализатор пакетов tcpdump

```
lumpy@ubuntu:~$ tcpdump -i wlp2s0 port 53
```

```
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
```

```
listening on wlp2s0, link-type EN10MB (Ethernet), capture size 65535 bytes
```

```
... ... ... ...
```

pts/1

```
lumpy@ubuntu:~$ host 2x2tv.ru
```

```
2x2tv.ru ① has address 146.158.12.222 ②
```

```
2x2tv.ru mail is handled by 10 fm.tnt-tv.ru.
```

pts/2

```
15:11:32.139394 IP ubuntu.local.40694 > 192.168.100.1.domain: 2656+ [1au] A? 2x2tv.ru. (37)
```

```
15:11:32.144674 IP 192.168.100.1.domain > ubuntu.local.40694: 2656 1/0/1 A 146.158.12.222 (53)
```

```
15:11:32.145260 IP ubuntu.local.22022 > 192.168.100.1.domain: 63760+ [1au] MX? 2x2tv.ru. (37)
```

```
15:11:32.147959 IP 192.168.100.1.domain > ubuntu.local.48132: 63760 1/0/1 MX fm.tnt-tv.ru. 10 (63)
```

```
... ... ... ...
```

pts/1

^C

4 packets captured

4 packets received by filter

0 packets dropped by kernel

Анализу подвергается работа DNS-клиента host, отображающего имя домена 2x2tv.ru на IP-адрес и имена его почтовых адресов (MX-записи DNS).

В результате анализа захваченных пакетов наблюдаются запросы и ответы DNS-протокола к локальному кэширующему серверу 192.168.106.1, который на запрос A? адреса IPv4, соответствующего имени 2x2tv.ru, отвечает адресом A 146.158.12.222.

Терминальный анализатор пакетов tshark, позволяющий проводить детальный анализ прикладных протоколов, таких как SSH, HTTP, FTP, NFS и пр., проиллюстрирован в листинге 6.36.

(Гораздо удобнее, конечно, использовать его графический вариант – wireshark)

Здесь анализируется работа пользовательского агента curl, запрашивающего Web-ресурс по адресу <http://ipinfo.io/city>.

В результате захвата пакетов на интерфейсе (-i, interface) wlp2s0, адресованных порту port 80, просматриваются (-R, read filter) пакеты, содержащие http-запросы, при этом детальному анализу (-V, view) подвергается только (-o, only) их http-содержимое.

В результате анализа, например, можно сделать вывод о программном обеспечении Web-сервера, обслуживающего сайт <http://ipinfi.io>.

Листинг 6.36. Анализатор пакетов wireshark

```
lumpy@ubuntu:~$ tshark -i wlp2s0 -V -O http,data-text-lines -Y http port 80
```

```
Capturing on wlp2s0
```

```
... ... ... ...
```

```
pts/2
```

```
lumpy@ubuntu:~$ curl http://ipinfo.io/city
```

```
Saint Petersburg
```

```
pts/1
```

```
... ... ... ...
```

```
Frame 4: 131 bytes on wire (1048 bits), 131 bytes captured (1048 bits) on interface 0
```

```
Ethernet II, Src: PcsCompu_a9:78:36 (08:00:27:a9:78:36), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
```

```
Internet Protocol Version 4, Src: 10.0.2.15, Dst: 216.239.36.21
```

```
Transmission Control Protocol, Src Port: 38534, Dst Port: 80, Seq: 1, Ack: 1, Len: 77
```

```
Hypertext Transfer Protocol
```

```
GET /city HTTP/1.1\r\n
```

```
[Expert Info (Chat/Sequence): GET /city HTTP/1.1\r\n]
```

```
[GET /city HTTP/1.1\r\n]
```

```
[Severity level: Chat]
```

```
[Group: Sequence]
```

```
Request Method: GET
```

```
Request URI: /city
```

```
Request Version: HTTP/1.1
```

Host: ipinfo.io\r\nUser-Agent: curl/7.65.3\r\nAccept: */*\r\n\r\n

[Full request URI: http://ipinfo.io/city]

[HTTP request 1/1]

...

Frame 6: 441 bytes on wire (3528 bits), 441 bytes captured (3528 bits) on interface 0
Ethernet II, Src: RealtekU_12:35:02 (52:54:00:12:35:02), Dst: PcsCompu_a9:78:36 (08:00:27:a9:78:36)
Internet Protocol Version 4, Src: 216.239.36.21, Dst: 10.0.2.15
Transmission Control Protocol, Src Port: 80, Dst Port: 38534, Seq: 1, Ack: 78, Len: 387
Hypertext Transfer Protocol

HTTP/1.1 200 OK\r\n

[Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]

[HTTP/1.1 200 OK\r\n]

[Severity level: Chat]

[Group: Sequence]

Response Version: HTTP/1.1

Status Code: 200

[Status Code Description: OK]

Response Phrase: OK

Date: Sat, 23 Nov 2019 23:27:18 GMT\r\n

Content-Type: text/html; charset=utf-8\r\n

Content-Length: 7\r\n

[Content Length: 7]

x-cloud-trace-context: 8fa415d163cf0496576a136652cac2f6b7935058721407822980\r\n

Access-Control-Allow-Origin: *\r\n

X-Frame-Options: DENY\r\n

X-XSS-Protection: 1; mode=block\r\n

X-Content-Type-Options: nosniff\r\n

Referrer-Policy: strict-origin-when-cross-origin\r\n

Via: 1.1 google\r\n

\r\n

[HTTP response 1/1]

[Time since request: 0.145039080 seconds]

[Request in frame: 4]

[Request URI: http://ipinfo.io/city]

File Data: 7 bytes

Line-based text data: text/html (1 lines)

Saint Petersburg\r\n

^C2 packets captured

Сетевой сканер nmap

Сетевой сканер W:[nmap] предназначен для поиска служб по их открытым портам на указанных узлах сети.

В примере из листинга 6.37 показан процесс и результаты сканирования узла 192.168.0.1 (беспроводной маршрутизатор, арендованный у провайдера Интернета).

Сканирование выполнялось способом TCP connect scan, т. е. предпринималась попытка установить соединение TCP с каждым из 1000 «популярных» портов.

В результате оказывается, что на узле открыты 4 порта, доступные для присоединения.

Листинг 6.37 Сетевой сканер nmap

```
lumpy@ubuntu:~$ nmap -p -vvv --reason 192.168.0.1
Starting Nmap 7.80 ( https://nmap.org ) at 2019-11-24 11:46 MSK
Initiating Ping Scan at 11:46
Scanning 192.168.0.1 [2 ports]
Completed Ping Scan at 11:46, 0.01s elapsed (1 total hosts)
Initiating Connect Scan at 11:46
Scanning 192.168.0.1 [1000 ports]
Discovered open port 80/tcp on 192.168.0.1
Discovered open port 22/tcp on 192.168.0.1
Discovered open port 1900/tcp on 192.168.0.1
Discovered open port 49152/tcp on 192.168.0.1
Completed Connect Scan at 11:46, 0.57s elapsed (1000 total ports)
```

Nmap scan report for 192.168.0.1

Host is up, received syn-ack (0.054s latency).

Scanned at 2019-11-24 11:46:18 MSK for 1s

Not shown: 996 closed ports

Reason: 996 conn-refused

PORT	STATE	SERVICE	REASON
22/tcp	open	ssh	syn-ack
80/tcp	open	http	syn-ack
1900/tcp	open	upnp	syn-ack
49152/tcp	open	unknown	syn-ack

Мониторинг сетевых соединений процессов

Интерфейс сокетов в целом является продолжением идеи «файлов», специально предназначенных для межпроцессного взаимодействия, некоторым развитием «файловой» природы простейших именованных и неименованных каналов.

Таким образом, сокеты сетевых семейств `ip`, `ipv6` и прочие обладают «файловыми» свойствами точно так же, как и локальные сокеты `unix`.

Например, каждый сетевой сокет идентифицируется при помощи файлового (!) дескриптора в таблице открытых файлов процесса, что проиллюстрировано в листинге 6.38 при помощи `lsof` и `ss`.

Листинг 6.38. Файловые дескрипторы сетевых сокетов

```
lumpy@ubuntu:~$ pgrep lftp
```

```
6056
```

```
lumpy@ubuntu:~$ lsof -i 4 -a -p 6056
```

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE/OFF	NODE	NAME
---------	-----	------	----	------	--------	----------	------	------

lftp	6056	lumpy	4	IPv4	88150	0t0	TCP	ubuntu.local:43578->napoleon.ftp.acc.umu.se:ftp (ESTABLISHED)
------	------	-------	---	------	-------	-----	-----	---

```
lumpy@ubuntu:~$ ss -p port = :ftp
```

Netid	State	Recv-Q	Send-Q	Local Address:Port	Peer Address:Port
-------	-------	--------	--------	--------------------	-------------------

tcp	ESTAB	0	0	192.168.0.101:43578	194.71.11.173:ftp users:(("lftp",pid=6056,fd=4))
-----	-------	---	---	---------------------	--

Более детально проследить за жизненным циклом сетевого сокета позволяет трассировка «сокетных» системных вызовов socket, connect, bind, listen, accept), send и recv.

В примере из листинга 6.39 показана трассировка «клиентского» сокета пользовательского агента curl, загружающего Web-ресурс <http://www.gnu.org/graphics/agnuheadern-xtern.txt>.

Системный вызов `socket` создает потоковый сокет семейства `ip`, которому назначается первый свободный файловый дескриптор `FD = 3`, после чего системный вызов `connect` инициирует установку соединения этого сокета с портом 80 узла 209.51.188.148.

После установки соединения системный вызов `sendto` отсылает Web-серверу команду W:[HTTP] протокола GET на получение запрашиваемого ресурса, а несколько системных вызовов `recvfrom` получают запрошенный ресурс.

Листинг 6.39. Файловые дескрипторы сетевых сокетов socket, connect,sendto и recvfrom

```
lshru@ubuntu:~$ strace -f -e trace=network curl http://www.gnu.org/graphics/agnuheadterm-xterm.txt
...
socket(AF_INET, SOCK_STREAM, IPPROTO_TCP) = 3
setsockopt(3, SOL_TCP, TCP_NODELAY, [1], 4) = 0
...
connect(3, {sa_family=AF_INET, sin_port=htons(80), sin_addr=inet_addr("209.51.188.148")}, 16) = -1
EINPROGRESS (Операция выполняется в данный момент)
...
sendto(3, "GET /graphics/agnuheadterm-xterm"..., 106, MSG_NOSIGNAL, NULL, 0) = 106
recvfrom(3, "HTTP/1.1 200 OK\r\nDate: Sun, 24 N"..., 102400, 0, NULL, NULL) = 1382
recvfrom(3, "249m\342\226\204\33[38;5;246m\342\226"..., 17588, 0, NULL, NULL) = 12438
...
...
```

Жизненный цикл «серверных» сокетов чуть более сложен и предполагает использование одного «слушающего» сокета для клиентских подключений и по одному «обслуживающему» сокету на каждого подключенного клиента.

В примере из листинга 6.40 показана трассировка простейшего Web-сервера, реализованного модулем `SimpleHTTPServer` языка программирования Python, Web- сервер запускается из «рабочего» каталога `/usr/share/doc` и предоставляет Web- доступ ко всем файлам этого каталога, используя «нестандартный» порт 8000.

Для начала при помощи `socket` создается потоковый сокет семейства `ip`, которому назначается первый свободный файловый дескриптор `FD = 3`.

Этот сокет и будет выступать в роли «слушающего», т. е. принимающего клиентские соединения, поэтому ему «привязывается» адрес `0.0.0.0` и порт `8000` (куда и будут поступать клиентские соединения) при помощи системного вызова `bind`, а сам сокет переводится в слушающее состояние системным вызовом `listen`.

Все входящие клиентские соединения ставятся в очередь «слушающего» сокета и изымаются из нее системным вызовом accept, который создает для каждого клиентского соединения собственный сокет (клонируя слушающий).

При поступлении клиентского соединения был создан новый «обслуживающий» сокет с файловым дескриптором FD = 4, используя который при помощи recv была получена W:[HTTP]-команда GET на доступ к «корневому» ресурсу сервера, а с помощью нескольких системных вызовов send в ответ был направлен сформированный HTML-список файлов в каталоге /usr/share/doc.

Листинг 6.40. Сетевой сервер: системные вызовы socket, bind, listen, accept, send и recv.

```
lumpy@ubuntu:~$ cd /usr/share/doc
lumpy@ubuntu:/usr/share/doc$ strace -fe trace=network python -m SimpleHTTPServer
socket(AF_INET, SOCK_STREAM, IPPROTO_IP) = 3
setsockopt(3, SOL_SOCKET, SO_REUSEADDR, [1], 4) = 0
bind(3, {sa_family=AF_INET, sin_port=htons(8000), sin_addr=inet_addr("0.0.0.0")}, 16) = 0
            ...
listen(3, 5)                                = 0
            ...
Serving HTTP on 0.0.0.0 port 8000 ...
accept(3, {sa_family=AF_INET, sin_port=htons(55094), sin_addr=inet_addr("127.0.0.1")}, [16]) = 4
recvfrom(4, "GET / HTTP/1.1\r\nHost: localhost:..., 8192, 0, NULL, NULL) = 78
127.0.0.1 - - [24/Nov/2019 12:14:54] "GET / HTTP/1.1" 200 -
sendto(4, "HTTP/1.0 200 OK\r\n", 17, 0, NULL, 0) = 17
sendto(4, "Server: SimpleHTTP/0.6 Python/2."..., 41, 0, NULL, 0) = 41
sendto(4, "Date: Sun, 24 Nov 2019 09:14:54 "..., 37, 0, NULL, 0) = 37
sendto(4, "Content-type: text/html; charset"..., 40, 0, NULL, 0) = 40
sendto(4, "Content-Length: 86602\r\n", 23, 0, NULL, 0) = 23
sendto(4, "\r\n", 2, 0, NULL, 0)                = 2
sendto(4, "<!DOCTYPE html PUBLIC "-//W3C//D"..., 8192, 0, NULL, 0) = 8192
            ...
shutdown(4, SHUT_WR)                         = 0xC
```

```
lumpy@ubuntu:~$ wget http://ubuntu:8000
--2019-11-24 12:18:05--  http://ubuntu:8000/
Распознаётся ubuntu (ubuntu)... 127.0.1.1
Подключение к ubuntu (ubuntu)|127.0.1.1|:8000... соединение установлено.
НПТР-запрос отправлен. Ожидание ответа... 200 OK
Длина: 86602 (85K) [text/html]
Сохранение в: «index.html»

100%[=====] 84,57K      --.-K/s   за 0,002s

2019-11-24 12:18:05 (54,6 MB/s) - «index.html» сохранён [86602/86602]
```

Сетевая подсистема ОС Linux чрезвычайно развита на всех ее уровнях — от сетевых интерфейсов и протоколов и до прикладных сетевых служб. На сегодняшний день колоссальное количество сетевых устройств работают под управлением.

Linux — маршрутизаторы, сетевые хранилища, медиаплееры, TV-боксы, планшеты, смартфоны и прочие «встраиваемые» и мобильные устройства.

К сожалению, рассмотреть весь пласт сетевых возможностей в рамках этой лекции не представляется возможным, т. к. потребует от студентов серьезного понимания устройства и функционирования самих сетевых протоколов стека TCP/IP, что не является предметом настоящего рассмотрения.

Основополагающим результатом текущей главы должно стать понимание принципов организации сетевого взаимодействия в Linux, необходимое и достаточное в качестве базы для последующего самостоятельного расширенного и углубленного изучения.

Не менее полезными в практике администратора и программиста будут навыки использования инструментов трассировки и мониторинга сетевых сокетов, а. в особенно «непонятных» ситуациях — навыки применения анализаторов пакетов.

Исключительное место (эдакий «швейцарский нож») среди прочих сетевых инструментов Linux займет служба SSH, применение которой найдется в дальнейшем материале, при распределенном использовании оконной системы X Window System, являющейся основой современного графического интерфейса пользователя.

ОС Аврора 4
QP ОС
Google Fuchsia OS

Лекция 16 +

Google Fuchsia OS и ее компоненты

Fuchsia — операционная система, разрабатываемая корпорацией Google.

Основа системы — собственное микроядро Zircon, не Linux.

Является небольшой ОС, предназначеннной для встраиваемых систем, разработанная Тревисом Гейсельбрехтом, создателем ядра NewOS

Система базируется на микроядре Zircon, основанном на наработках проекта LK, расширенного для применения на различных классах устройств, включая смартфоны и персональные компьютеры.

Zircon расширяет LK поддержкой процессов и разделяемых библиотек, уровнем пользователя, системой обработки объектов и моделью обеспечения безопасности на основе capability.

Драйверы реализуются в виде работающих в пространстве пользователя динамических библиотек, загружаемых процессом devhost и управляемых менеджером устройств (devmg, Device Manager).

Разработчики реализовали драйверы как работающие в пространстве пользователя динамические библиотеки.

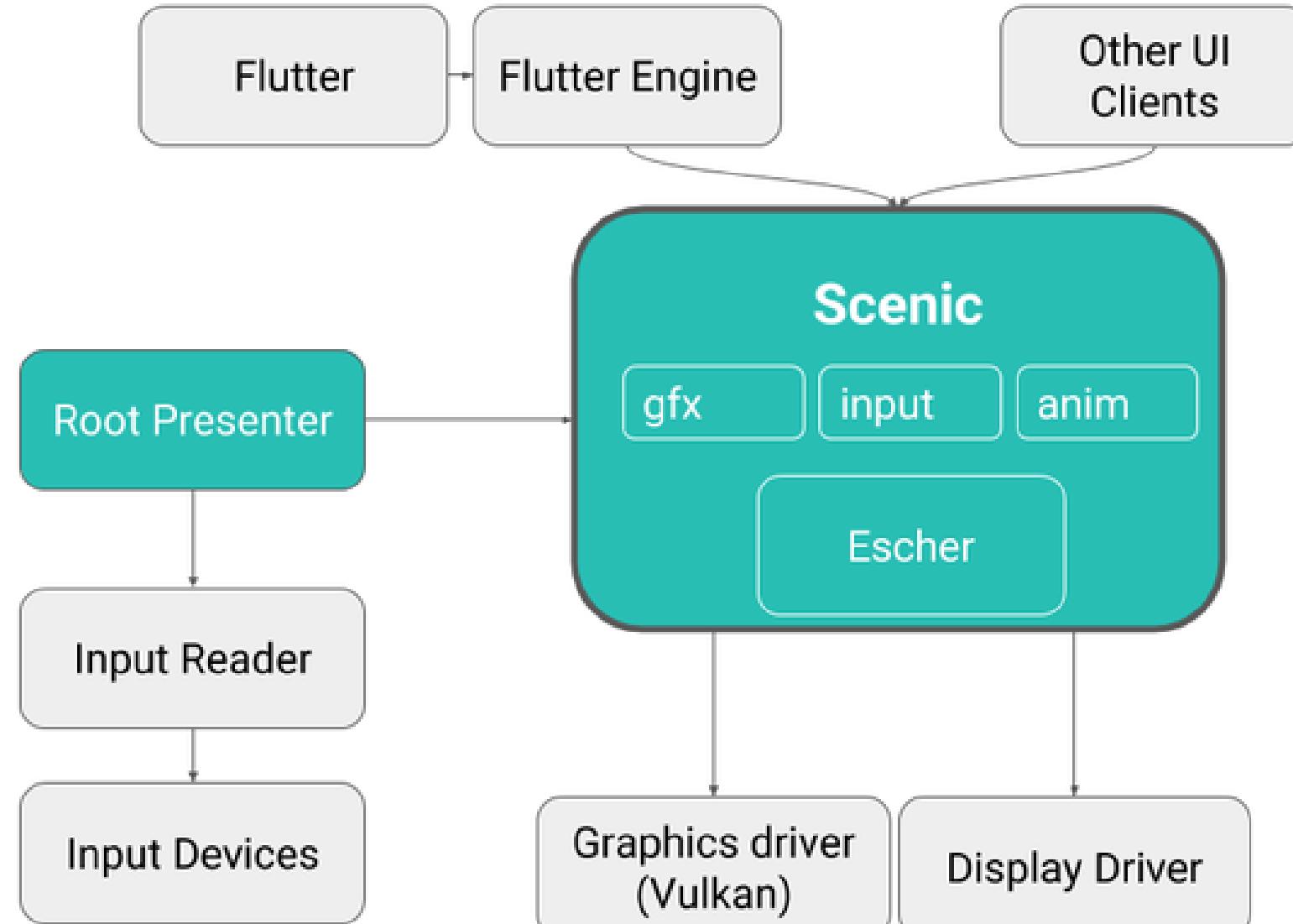
Загружаются они при помощи процесса devhost, а управляются менеджером устройств (devmg, Device Manager).

Пользовательская оболочка операционной системы, Armadillo, написана на языке Dart с использованием фреймворка Flutter.

Проект также включает и развивает:

- фреймворк для построения интерфейсов пользователя Peridot;
- пакетный менеджер Fargo;
- стандартную библиотеку libc;
- систему рендеринга Escher;
- Vulkan-драйвер Magma;
- композитный менеджер Scenic;
- файловые системы MinFS, MemFS, ThinFS (FAT на языке Go) и Blobfs
- менеджер разделов FVM.

Для разработки приложений предоставляется поддержка языков C/C++, Dart, в системных компонентах также допускается использование Rust, в сетевом стеке — Go, а в системе сборки языка — Python.



Структурная схема взаимодействия системы

В процессе загрузки используется системный менеджер, включающий arptmgr для создания начального программного окружения, sysmgr для формирования загрузочного окружения и basemgr для настройки пользовательского окружения и организации входа в систему.

Для обеспечения безопасности предлагается продвинутая система sandbox-изоляции, в которой новые процессы не имеют доступа к объектам ядра, не могут выделять память и не могут запускать код, а для доступа к ресурсам применяется система пространств имён, определяющая доступные полномочия.

Платформа предоставляет фреймворк для создания компонентов, представляющих собой программы, запускаемые в своём sandbox, которые могут взаимодействовать с другими компонентами через IPC.

Fuchsia OS — полностью открытая операционная система

Большой плюс операционной системы в том, что она открыта — корпорация изменила модель позиционирования платформы в 2020 году.

Соответственно, патчи и коммиты разработчики принимают от всех желающих.

После открытия Fuchsia для сообщества коммиты стал принимать управляющий совет, в состав которого вошла группа опытных технических руководителей компании.

Совет следит за выполнением дорожной карты проекта и администрирует пользовательские изменения.

Но и до изменения лицензии разработка ОС была полностью прозрачной — в течение четырех лет любой желающий мог оценивать изменения в репо проекта.

Разработчики позиционируют систему как безопасную и обновляемую, позиционируя ее как мультиплатформенную. Она может работать на ПК, умных телевизорах, колонках и прочих гаджетах.

QP ОС

Авторы и разработчики хранят свои тайны, и не хотят, чтобы вся их работа утекла в сеть и стала достоянием общественности.

В связи с этим, тестирование QP ОС возможно лишь по договору, и, на данный момент времени, только для юридических лиц.

Создатели данной ОС НТП «Криптософт».

Внутренняя структура ядра разбита на слои по безопасности и в целом подобна ядру Windows.

Формат исполняемых файлов собственный-СМФ(за исключением .Net приложений, для которых поддерживается PE)

Для графики разработана собственная библиотека GUIDO, которая воспринимается как перелицованный винда.

Но здесь все в порядке. Это сделано заново

Состав ядра примерно такой:

- Диспетчер задач (написан заново).
- Менеджер памяти (написан заново).
- Драйверы файловых систем (написаны заново)
- Сетевые стеки(написаны заново)
- Система безопасности(написана заново)
- Визуальная подсистема (написана заново)
- Графическая система (написана заново)

Аппаратная совместимость

- Поддержка ACPI и UEFI
- До 256 ядер процессоров
- До 9 Тбайт RAM
- IDE, SATA, SCSI, RAID, iSCSI, FC
- USB 3.1
- IEEE 802.3 802.11

В состав PQ ОС, со слов разработчика, входит следующее ПО

- FTP-сервер
- SMB сервер/клиент
- Web-server
- Nginx
- QP VMM
- Почтовый сервер
- Почтовый клиент
- Игры
- Браузер
- Офис
- DNS-сервер
- RDP клиент и RDP сервер
- Скриншот содержимого каталога «программы»:

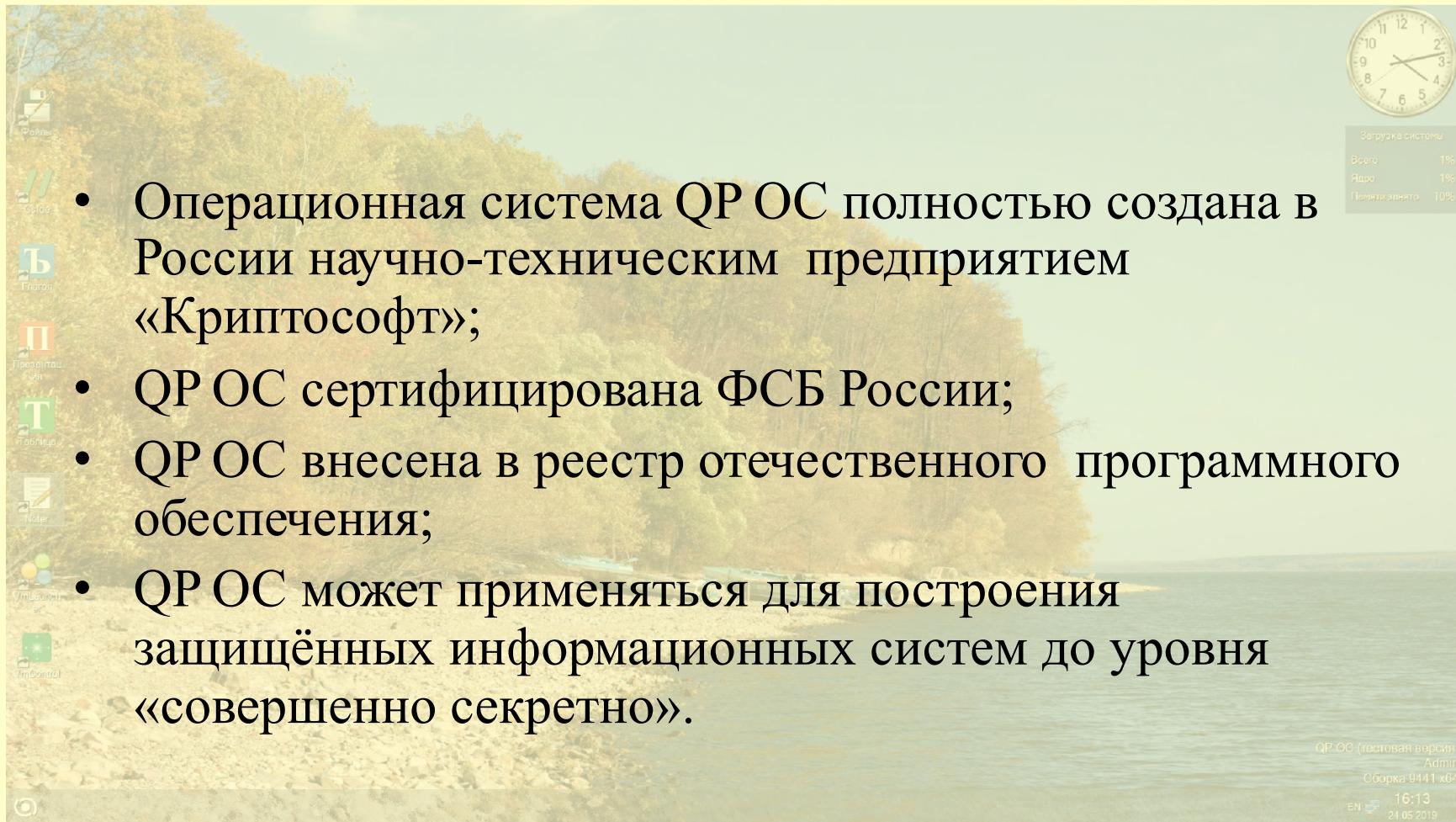
Преимущества использования полностью отечественной операционной системы

- Все исходные коды разрабатываются в России;
- Для каждого компонента программного обеспечения имеются компетенции, позволяющие модифицировать исходный код;
- НТП «Криптософт» самостоятельно разрабатывает код создаваемых информационных систем.



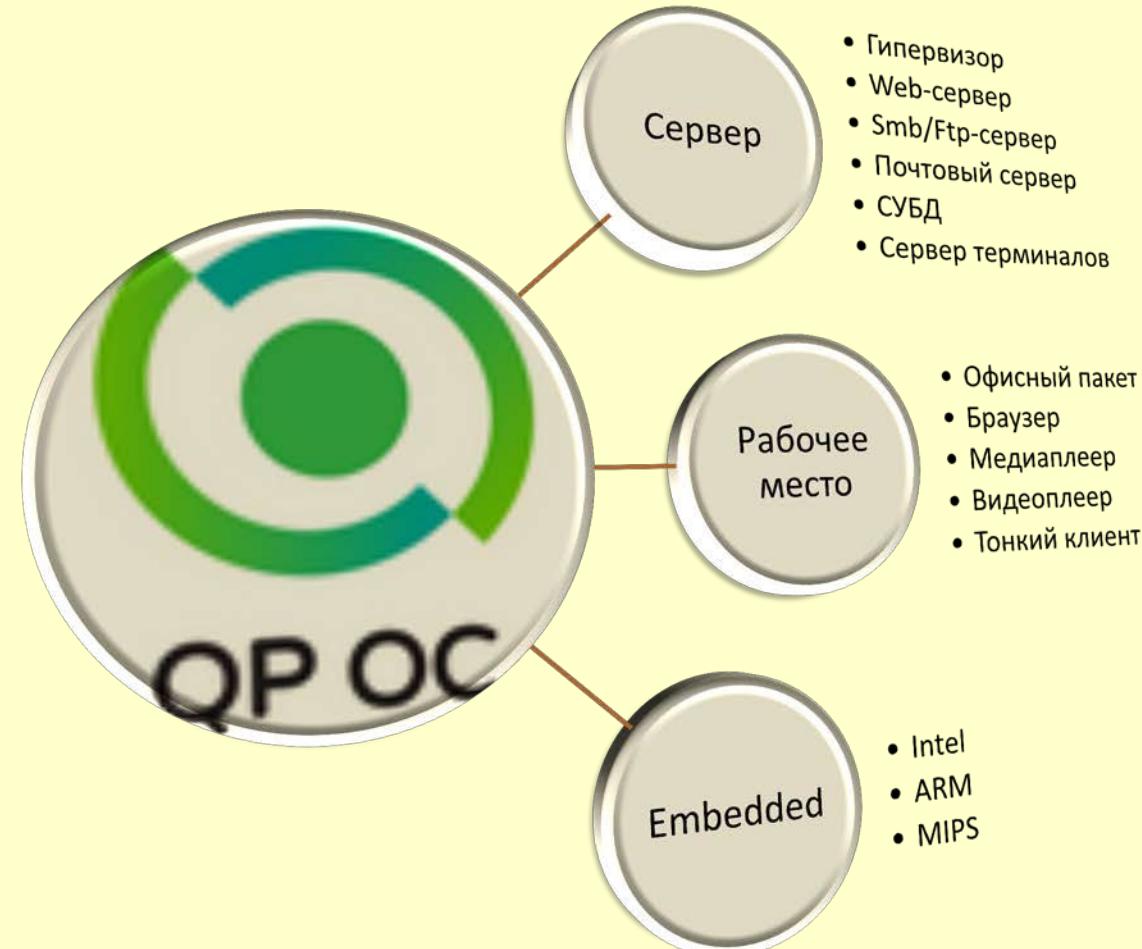


QP ОС





Применение QP OC





Полный комплекс механизмов безопасности



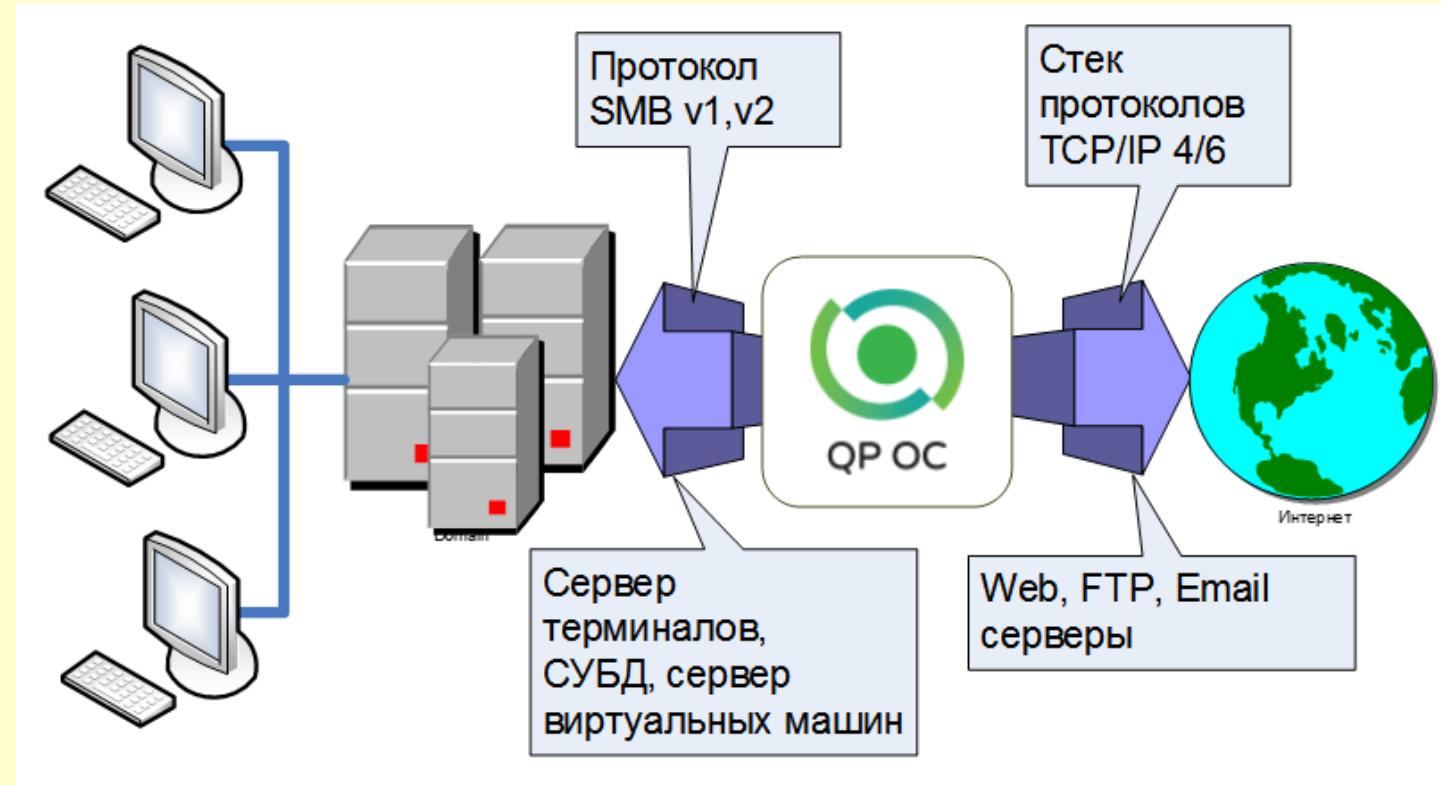


Серверные компоненты в составе QP ОС





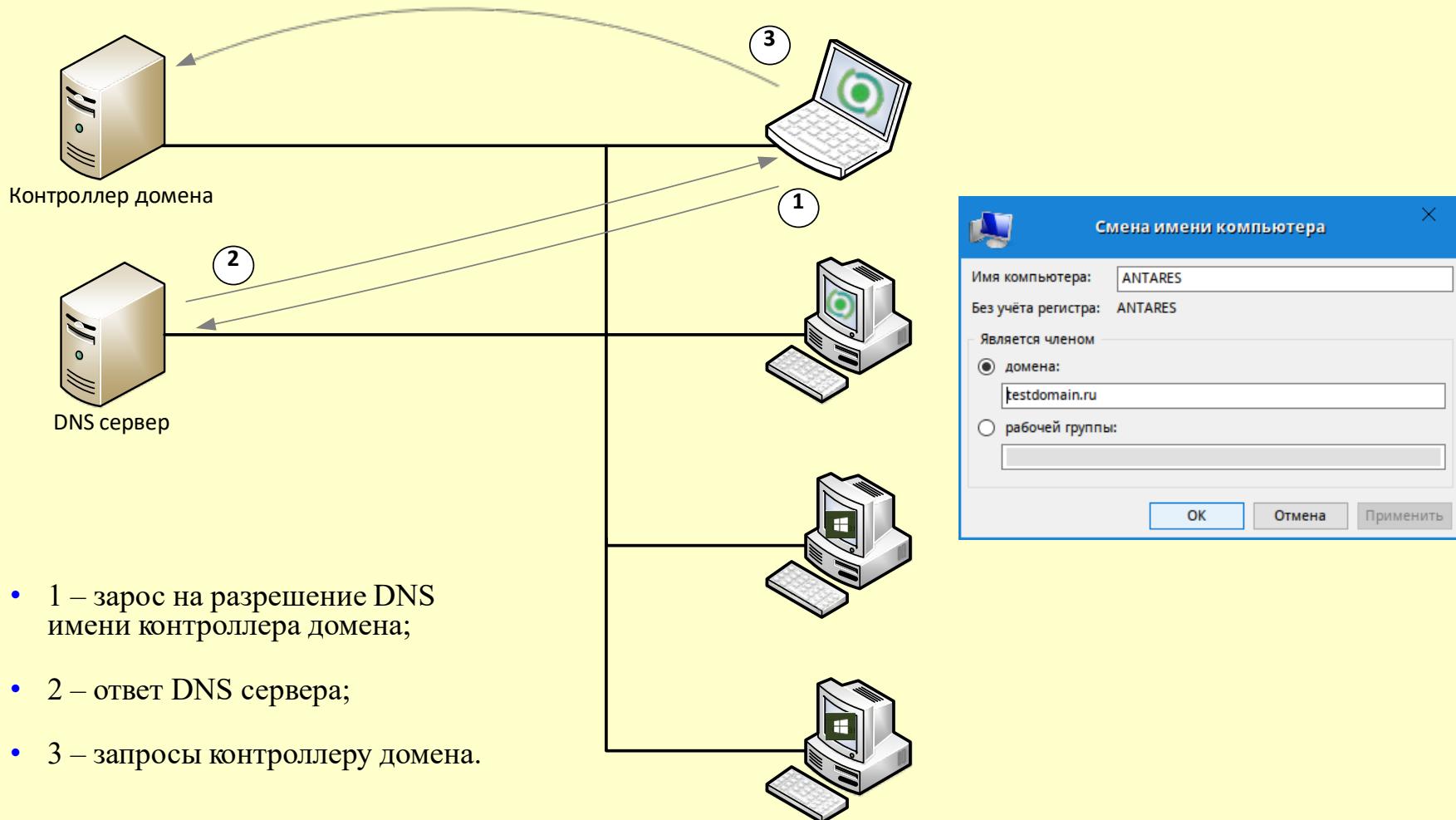
Сетевые возможности QP ОС



- Все сетевые протоколы в QP ОС реализованы заново.



Подключение к доменам безопасности AD





Пакет офисных приложений



В состав офисного пакета входят следующие приложения

- Редактор текстовых документов Глагол;
- Редактор таблиц Таблица;
- Менеджер презентаций Презентация;
- Клиент электронной почты.





Глагол



Файл Главная Разметка страницы Вставка

дипломTOC.docx - Глаголь

Поля Ориентация Размер Параметры страницы Абзац

Отступ Интервал

Слева: 0 см До: 0 см
Справа: 0 см После: 0 см

Интервал

Абзац

Исходя из вышеописанного, следует, что в ходе выполнения выпускной квалификационной работы был разработан работоспособный программный модуль, отвечающий требованиям технического задания.

Список использованных источников

1. Вийера, Р. Программирование баз данных Microsoft SQL Server 2005: базовый курс / Р. Вийера ; пер. с англ. – М. : ООО «И.Д.Вильямс», 2010.
2. Гарсия-Молина, Г., Ульман, Д. Д., Уидом, Д. Системы баз данных : Полный курс/ Г. Гарсия-Молина, Д.Д. Ульман, Д. Уидом ; пер. с англ. – М. : Издательский дом «Вильямс», 2012.
3. Фролов, К.М. Оптимизация доступа к данным на основе индексов / И.А. Казакова, К.М. Фролов // Мир современной науки №3 (25). – Москва, 2014 – С. 32-34.
4. Ахо, А.В. Структуры данных и алгоритмы / А.В.Ахо, Д.Хопкрофт,

Страница 43 из 43 100%



Глагол



diplomTOC.docx - Глаголь

Файл Главная Разметка страницы Вставка Таблица

Команды Стили

Добавить параграф перед таблицей
Добавить параграф после таблицы

Вставить Удалить

Колонку Строку Колонку Строку Таблицу

	постоянной памяти			
15	Проектирование завершено	Веха	13,14	-
16	Реализация	Фаза		-
17	Реализация В-дерева в оперативной памяти	Задача	13	2
18	Реализация В-дерева в постоянной памяти	Задача	14	2
19	Реализация завершена	Веха	17,18	-
20	Контроль качества ПО	Фаза		-
21	Модульное тестирование В-дерева в оперативной памяти	Задача	17	2
22	Модульное тестирование В-дерева в постоянной памяти	Задача	18	2
23	Оценка качества проведенного тестирования	Задача	21,22	1

Страница 34 из 38

6

100%



Глагол



Рисунок 4 – Общая структура хранилища СУБД

Данная схема справедлива для индексов, работающих с постоянной памятью. Индексы, взаимодействующие с оперативной памятью, работают с записями напрямую [6].

1.1.2 Анализ готовых решений

В качестве готовых решений будет разумным рассмотреть два наиболее популярные СУБД – SQL Server 2012 компании Microsoft и PostgreSQL.

СУБД SQL Server 2012 имеет закрытый исходный код и предоставляет пользователю весь обширный спектр работы с базами данных. Данная СУБД поддерживает различные языки запросов, что облегчает работу

The screenshot shows a Microsoft Word document window titled 'diplomTOC.docx - Глаголь'. The ribbon tabs are 'Файл', 'Главная', 'Разметка страницы', and 'Вставка'. The 'Главная' tab is selected. The ribbon has several groups: 'Шрифт' (Font) with 'Times New Roman' and size '14'; 'Абзац' (Paragraph) with alignment and spacing options; 'Стили' (Styles) showing 'Обычный' (Normal) and 'Заголовок 1' (Header 1); 'Создать стиль' (Create Style); and 'Найти' (Find). The main content area contains a diagram of a database structure. It features a large rectangle labeled 'Индексная страница' (Index page) at the top and 'Запись' (Record) below it. Above the rectangle, there is a triangular pointer pointing towards it. On the left side of the content area, there is a vertical ruler with numbers from 7 to 19. At the bottom left, it says 'Страница 6 из 38'. At the bottom right, it says '100%'. The entire screenshot is set against a light yellow background.



Таблица



Образец заполнения справки 2-НДФЛ.xlsx - Таблица

Файл Главная Вставка Формула Макет

Пересчитать всё
Пересчитать текущий
Имена
Вычисления
Определения имен

AF47 : =Q23+Q24+Q25+Q26+Q27+Q28+Q29+Q30+Q31+Q32+Q33+Q34+Q35+Q36+Q37+BW23+BW24

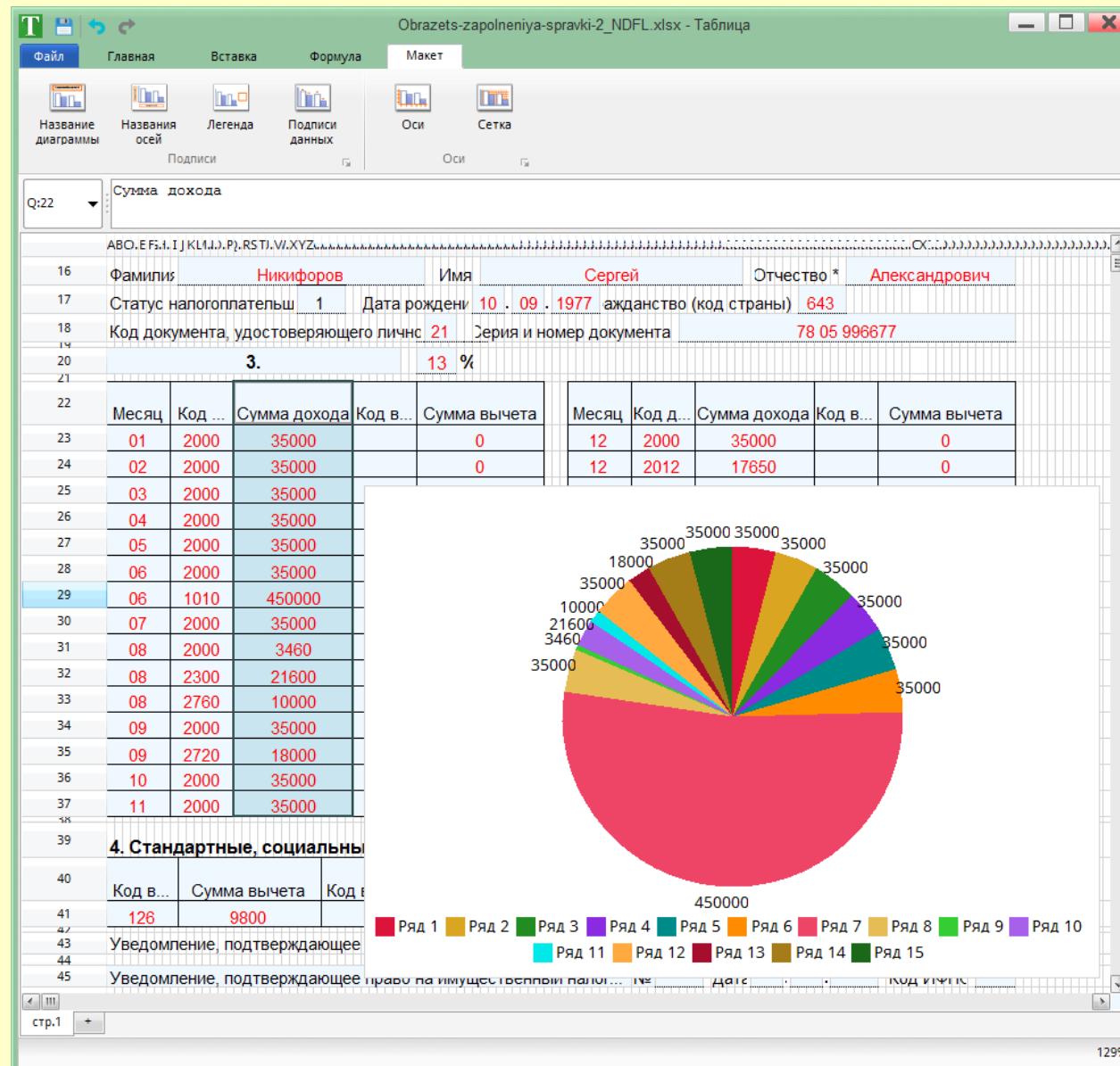
22	Месяц	Код ...	Сумма дохода	Код в...	Сумма вычета	Месяц	Код д...	Сумма дохода	Код в...	Сумма вычета
23	01	2000	35000		0	12	2000	35000		0
24	02	2000	35000		0	12	2012	17650		0
25	03	2000	35000		0					
26	04	2000	35000		0					
27	05	2000	35000		0					
28	06	2000	35000		0					
29	06	1010	450000		0					
30	07	2000	35000		0					
31	08	2000	3460		0					
32	08	2300	21600		0					
33	08	2760	10000	503	4000					
34	09	2000	35000		0					
35	09	2720	18000	501	4000					
36	10	2000	35000		0					
37	11	2000	35000		0					
39	4. Стандартные, социальные и имущественные налоговые вычеты									
40	Код в...	Сумма вычета	Код в...	Сумма вычета	Код в...	Сумма вычета	Код в...	Сумма вычета	Код в...	Сумма вычета
41	126	9800		0		0		0		0
42	Уведомление, подтверждающее право на социальный налоговый вычет № _____ Дата: _____. Код ИФНС: _____									
43	Уведомление, подтверждающее право на имущественный налоговый вычет № _____ Дата: _____. Код ИФНС: _____									
46	5. Общие суммы дохода и налога									
47	Общая сумма дохода	905710	Сумма налога удержанная	115428						
48	Налоговая база	887910	Сумма налога перечисленная	115428						
49	Сумма налога исчисленная	115428,3	Сумма налога, излишне удержанная ...	0						

стр.1

129%



Таблица





Презентация



qpos12.pptx* - Презентация

Файл Главная Вставка Дизайн Показ слайдов

Вставить Вырезать Копировать Создать слайд Arial 36 Шрифт Выровнять текст Абзац Найти

Буфер обмена Слайды Редактирование

Научно-техническое предприятие
КРИПТОСОФТ

Импортозамещение на основе полностью отечественной операционной системы

Валерий Егоров
НТП «Криптософт»

Слайд 1 из 12

72% 15



Презентация



qpos12.pptx* - Презентация

Файл Главная Вставка Дизайн Показ слайдов

Ориентация слайда Размер слайда

Параметры страницы

Темы

Цвета Фон

Скрыть фоновые рисунки

Формат фона

Слайд 5 из 6

Сетевые возможности QP ОС

Все сетевые протоколы в QP ОС реализованы заново.



Почта



Почта - Тестовое задание по C#

Файл Сообщения Вид Сервис

Отправить/Получить Создать | Письмо | Контакты | Папки | Помощь

Сообщения
Входящие
Исходящие
Отправленные
Удаленные

Входящие

Поиск...

Тема Автор
Сегодня
Тестовое задание по C# "Андрей Подгорный"

Тестовое задание по C#

Тема: Тестовое задание по C#
Автор: "Андрей Подгорный" <example@cryptosoft.ru>
Кому: <test@cryptosoft.ru>

Время отп.: 27.05.2019 11:15:51

Здравствуйте выполнил ваше задание редактор просматривает и редактирует файлы болле 4Гб но скорость работы зависит от настроек буфера эти настройки можно менять в зависимости от конфигурации ПК в главном меню -> настройки -> настройки буфера по умолчанию 5Мб т.к. у меня слабый компьютер архив с проектом прикреплён к письму файл на котором тестировалась работа программы размером 12.267Гб

TextEditor_v_1_3.zip (99 Кбайт)

Готово



Браузер



Интернет

Mail.Ru: почта, поиск в интернете, новости, игры

https://mail.ru/

Mail.ru Почта Мой Мир Одноклассники Игры Знакомства Новости Поиск Все проекты ▾ Регистрация Вход

Поиск в интернете Картинки Видео Ответы

Найти

Новости Спорт Авто Кино ...

СКР возбудил дело против чиновника, напавшего на журналиста

В РПЦ сравнили протесты из-за храма с расстрелом царской супруги Ксении Коломойский рассказал, из-за чего поругался с Порошенко Отставание российских городов от Москвы оказалось огромным Почему Волга в этом году так сильно обмелела и чем это грозит Зеленая волна: что изменят итоги выборов в Европарламент Курс биткоина обновил годовой максимум Леди 40-летнюю Ани Лорак раскритиковали за пластику (фото) Все аптеки Как убрать «капельсиновую корку» с бедер Кино Пропустившая спецвыпуск «Голоса» дочь Алсу высказалась о суде над Навальным

Установите на дом! ECOSVET Перейти

Установите у входа в дом! Светильники для дома, гаража, балкона. Установка без проводов и инструментов. Впервые в РФ

Магазин "Экосвет" ИП Жуковская А.А. ИНН 77090475

Игры Revelation РОЛЕВАЯ Warface ШУТЕР

Клиентские 13 Браузерные 30

+18 завтра +24 \$ 64.42 -0.05 € 72.11 -0.14 Водолей — можно добиться успеха в

Transferring data from r.miradx.net...



Браузер



Портал государственных услуг Российской Федерации

→ Портал государственных услуг Российской Федерации +

https://www.gosuslugi.ru/

Для граждан Пенза RUS

госуслуги Услуги Оплата Поддержка

Введите название услуги или ведомства

Рекомендуем для жителей Пензенской области

Проверка штрафов Получение загранпаспорта Родители и дети

Справка об отсутствии судимости Запись к врачу Восстановление документов

Опрос: как должен выглядеть сайт вашего города или района?

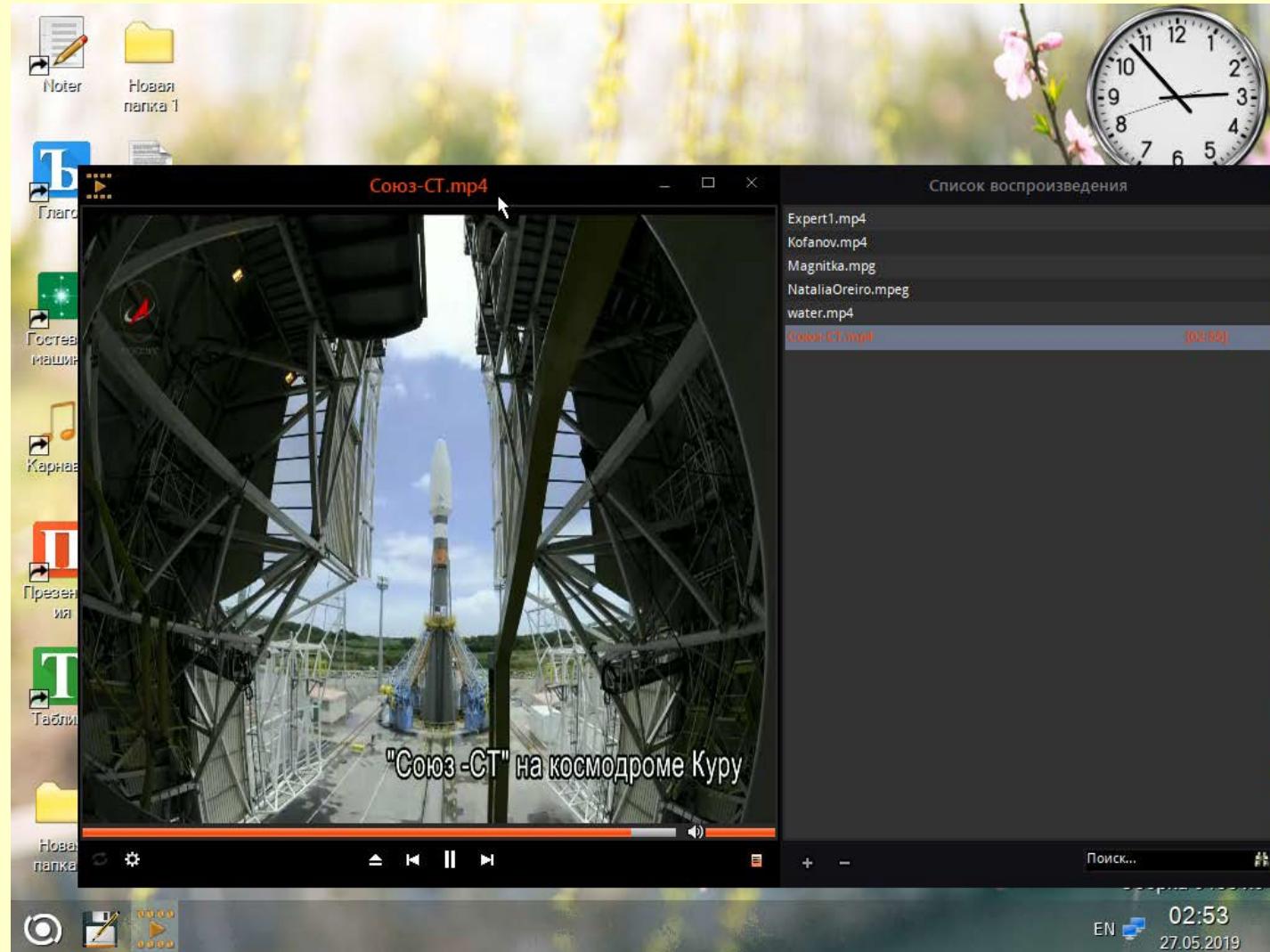
Помогите нам сделать государственные сайты лучше, это займет всего несколько минут

Зарегистрироваться Как зарегистрироваться

https://esia.gosuslugi.ru/registration



Видеоплеер





Для бесплатного тестирования
QP ОС обратитесь по адресу
qpos@cryptosoft.ru

ОС Аврора 4

Разработка ОС ведётся с 2016 года российской компанией «Открытая мобильная платформа» на базе Sailfish OS (с 2012 года разрабатывается финской Jolla), которую сильно модифицировали на уровне всех слоёв операционной системы:

- ядро и загрузчики (контроль файловой системы, проверка цифровых подписей при загрузке, в т. ч. отдельных модулей);
- системный слой (возможность удалённого управления, защита данных приложений, защита канала передачи данных, многопользовательский режим, поддержка токенов и многое др.);
- приложения (криптозаметки, браузер с поддержкой ГОСТ, средства работы с документами офисных форматов, и т.д.).

Мобильная ОС Sailfish, приобретенная в 2018 г. «Ростелекомом», отныне называется «Аврора».

Руководство компании считает новое название энергичным, позитивным, а главное, русскоязычным.

На данный момент это не единственная российская мобильная ОС.

Улучшения затронули все составляющие мобильной инфраструктуры Аврора 4:

- средства обеспечения безопасности,
- пользовательский интерфейс,
- встроенные приложения,
- инструменты администрирования парка устройств и средства для разработки приложений.

Возможности мобильной ОС

- Доверенная загрузка и контроль целостности загрузчика и файловой системы
- Встроенная верификация установки и запуска программ
- Встроенные политики безопасности
- Полный дистанционный контроль над всеми функциями смартфона
- Собственная платформа управления устройствами (ЕММ)
- Защита каналов связи (ГОСТ VPN)
- Многофакторная аутентификация (включая поддержку токенов)
- Шифрование данных
- Работа с электронной подписью (в том числе квалифицированной)

Основные функции и улучшения 4 версии

Унификация корпоративной и сертифицированной версий

Унифицированные механизмы безопасности в обеих версиях.

Механизмы безопасности, такие как

- подпись пакетов,
- валидация приложений,
- подпись модулей ядра,
- контроль целостности,
- шифрования пользовательских данных,
- изоляция приложений и другие

Теперь доступны в любой версии Авроры.

Переносимые приложения

Унифицированный (идентичный) исходный код приложений для корпоративной и сертифицированной версий

Единый SDK для обеих версий

Компиляция приложений под разные требования в едином инструменте.

Подпись пакетов для защиты программной среды от недоверенного и вредоносного ПО

Все приложения подписываются

Приложения подписываются для любой целевой версии Авроры — корпоративной и сертифицированной.

Единый ключ разработчика

Упрощение инфраструктуры подписи пакетов — единый ключ для подписи бинарных файлов и пакетов.

Настройка разрешенного к установке ПО

Спецификация разрешенных к установке приложений на основе избранных сертификатов разработчиков.

Многопользовательский режим

Поддержка до 6 пользователей и администратора

Возможность реализации посменного режима работы на одном устройстве. Пользовательские данные полностью независимы.

Модульная система первоначальной загрузки

Кастомизация первого старта ОС под проект

Возможность изменения процедуры начальной настройки без внесения изменений в ОС. Быстрый ввод в эксплуатацию.

Улучшение пользовательского опыта

Новая клавиатура с предиктивным вводом

Высокая скорость набора текста. Меньше ошибок при вводе. Новый стиль, новая система подсказок, новые возможности кастомизации. Предиктивный ввод и автокоррекция.

Новый шрифт

Улучшенная читаемость текстов.

Новые элементы интерфейса

Новые кнопки, закладки и уведомления. Поддержка новых жестов и действий на экране уведомлений.

Переработанный механизм Атмосфер

Поддержка аппаратного ускорения обработки графики. Автоматическая адаптация стандартных интерфейсов приложений под палитру фонового изображения. Поддержка сложных эффектов сглаживания и размытия.

Новые иконки и мелодии

Обновлённый дизайн ОС.

Новые возможности для разработчиков приложений

Добавлены новые функции и элементы создания пользовательского интерфейса.

Обновленные стандартные приложения

Офис на движке LibreOffice 7

Высокая скорость загрузки и вывода на экран документов распространённых офисных форматов. Поддержка документов со сложным форматированием.

Обновленные приложения Файлы, Заметки, Календарь и др.

Улучшенный пользовательский опыт, новые возможности.

Поддержка сертификатов в почтовом клиенте

Установление защищенного подключения и шифрация почтового трафика.

Браузер на современном веб-движке Gecko 60

Поддержка новой версии ГОСТ TLS, интеграция технологии WebRTC и аудио- видео- кодеков для коммуникаций в реальном времени (например, видеоконференцсвязи), интеграция технологии WebGL, улучшенная поддержка технологий HTML 5 и JavaScript.

Новые и усовершенствованные API

NFC API

Работа с NFC-картами и токенами. Поддержка интерфейсов pscs-lite и pksc#11.

WebView API

Создание приложений отображающих веб-контент.

Antivirus API

Для разработки антивирусных приложений на базе стандартных механизмов ОС.

VPN API

Для разработки VPN приложений на базе стандартных механизмов ОС

API управления жизненным циклом приложений

Поддержка интеграции с магазином Аврора Маркет и сторонними ЕММ.

API для работы с QR-кодами

Поддержка баркодов, QR-кодов, штрихкодов штрих-кодов и т. д. Генерация QR-кодов.

Crypto API

Для разработки криптопровайдеров на базе QCA на базе стандартных механизмов ОС.

Keystore API

Для хранения чувствительной информации в защищенном хранилище.

MDM API

Новые политики удаленного управления парком устройств.

Безопасность

Изоляция приложений

Запуск приложений в «песочницах», защита информации от сторонних процессов.

Гранулярный доступ приложений к ресурсам

Приложения обязаны запрашивать доступ ко всем используемым ресурсам (камера, микрофон и т. д.).

Шифрование пользовательского раздела

Защита пользовательских данных при утере устройства.

Обновление загрузчиков (на ряде устройств)

Возможность обновления не только приложений и операционной системы, но и загрузчиков устройств.

СледопытSSL

Обновление встроенного средства криптозащиты.

Динамический контроль целостности программной среды

Контроль целостности программной среды осуществляется не только при загрузке ОС, но и в процессе работы.

Другие важные нововведения

Новый компилятор gcc 8

С поддержкой стандарта C++ 17-й версии.

Поддержка релизов с длительным сроком поддержки

Авора ТЕЕ — интегрированная среда для исполнения кода в
безопасном (доверенном) режиме

Авора СДЗ — программно-аппаратное средство доверенной загрузки с
корнем доверия в кристалле.

Встроенные приложения в ОС Авроре

- Телефон, контакты, сообщения (SMS/MMS)
- Галерея, камера, часы
- Почта, календарь, калькулятор, заметки, погода
- Криптозаметки
- Документы (просмотр pdf, doc, docx, ppt, pptx, xls, xlsx)
- Диктофон, мультимедиа (аудио, радио)
- Браузер с поддержкой ГОСТ-шифрования

Доступные сторонние приложения

- Антивирусы
- Доверенные мессенджеры
- Системы электронного документооборота (СЭД)
- Системы управления персоналом
- Доверенные хранилища, в том числе облачные
- Навигационные приложения
- Системы распознавания (документы, банковские карты и номера автомобилей)
- Системы профессиональной радиосвязи
- Приложения для контроля технических осмотров
- Др.

Устройства с ОС Аврора

Планшеты:

- Aquarius Cmp NS 220R[3]
- Aquarius Cmp NS 208R
- F+ Life Tab Plus
- F+ R570
- MIG T8

Смартфоны:

- Aquarius CMP NS M11
- MIG C55
- Qtech QMP-M1-N
- Qtech QMP-M1-N IP
- INOI R7

Работа с реестром Windows

Общие сведения о реестре Windows

Реестр Windows (системный реестр) - это иерархическая (древовидная) база данных, содержащая записи, определяющие параметры и настройки операционных систем Microsoft Windows.

Реестр в том виде, как он выглядит при просмотре редактором реестра, формируется из данных, источниками которых являются файлы реестра и информация об оборудовании, собираемая в процессе загрузки.

В описании файлов реестра на английском языке используется термин "Hive". В некоторых работах его переводят на русский язык как "Улей". В документации от Microsoft этот термин переводится как "Куст".

Файлы реестра создаются в процессе установки операционной системы и хранятся в папке **%SystemRoot%\system32\config**(обычно C:\windows\system32\config). Для операционных систем Windows это файлы с именами

default
sam
security
software
system

.

В операционных системах Windows Vista/Windows 7/8/10, файлы реестра располагаются также в каталоге `\Windows\system32\config` и имеют такие же имена, однако в этих ОС добавился новый раздел реестра для хранения данных конфигурации загрузки (Boot Configuration Data) с именем **BCD00000000**.

Файл с данными этого раздела имеет имя **bcd** и находится в скрытой папке **Boot** активного раздела (раздела, с которого выполняется загрузка системы).

Обычно, при стандартной установке Windows 7, создается активный раздел небольшого размера (около 100 мегабайт), который скрыт от пользователя и содержит только служебные данные для загрузки системы – загрузочные записи, менеджер загрузки **bootmgr**, хранилище конфигурации загрузки BCD, файлы локализации и программы тестирования памяти . Расположение куста **bcd** зависит от того, как сконфигурирован загрузчик системы при ее установке, и может находиться на том же разделе, где и каталог Windows.

*Место расположения файлов реестра в любой версии Windows можно просмотреть с помощью редактора реестра. В разделе **HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\hivelist** хранится информация о всех кустах, включая пользовательские профили, со ссылками на их расположение в файловой системе Windows.*

В процессе загрузки система получает монопольный доступ к файлам реестра и, поэтому, их невозможно открыть для просмотра, скопировать, удалить или переименовать обычным образом. Для работы с содержимым системного реестра используется специальное программное обеспечение - редакторы реестра (REGEDIT.EXE, REGEDT32.EXE), являющиеся стандартными компонентами операционной системы. Для запуска редактора реестра можно использовать меню кнопки "Пуск"- "Выполнить" - regedit.exe



Редактор реестра



Реестр Редактор Вид Избранное Справка



Мой компьютер

- + HKEY_CLASSES_ROOT
- + HKEY_CURRENT_USER
- + HKEY_LOCAL_MACHINE
- + HKEY_USERS
- + HKEY_CURRENT_CONFIG

Имя	Тип
-----	-----

Мой компьютер

После старта редактора, в левой части основного окна вы видите список корневых разделов (root keys) реестра. Каждый корневой раздел может включать в себя вложенные разделы (subkeys) и параметры (value entries) или ключи реестра.

Основное назначение корневых разделов:

HKEY_CLASSES_ROOT (Общепринятое сокращенное обозначение HKCR) - Ассоциации между приложениями и расширениями файлов и информацию о зарегистрированных объектах COM и ActiveX.

HKEY_CURRENT_USER (HKCU)- Настройки для текущего пользователя (рабочий стол, личные папки, настройки приложений). Этот раздел представляет собой ссылку на раздел HKEY_USERS\Идентификатор пользователя (SID) в виде S-1-5-21-854245398-1035525444-...

SID - это уникальный номер, идентифицирующий учетную запись пользователя, группы или компьютера. Он присваивается учетной записи при создании каждого нового пользователя системы. Внутренние процессы Windows обращаются к учетным записям по их кодам SID, а не по именам пользователей или групп. Если удалить, а затем снова создать учетную запись с тем же самым именем пользователя, то предоставленные прежней учетной записи права и разрешения не сохранятся для новой учетной записи, так как их коды безопасности будут разными. Аббревиатура SID образована от Security ID.

Идентификатор SID представляет собой числовое значение переменной длины, формируемое из номера версии структуры SID, 48-битного кода агента идентификатора и переменного количества 32-битных кодов субагентов и/или относительных идентификаторов (Relative IDentifiers, RID). Код агента идентификатора определяет агент, выдавший SID, и обычно таким агентом является локальная операционная система или домен под управлением Windows. Коды субагентов идентифицируют попечителей, уполномоченных агентом, который выдал SID, а RID - дополнительный код для создания уникальных SID на основе общего базового SID.

Для идентификатора S-1-5-21-854245398-1035525444: 1000, номер версии равен 1, код агента идентификатора - 5, а далее следуют коды четырех субагентов. В Windows NT и старше, при установке системы, создается один фиксированный (код 21) и три генерируемых случайным образом (числа после "S-1-5-21") кода субагентов. Также в процессе установки создаются некоторые (одинаковые для всех систем) учетные записи, как например, учетная запись администратора, которая всегда имеет RID равный 500

Для просмотра соответствия SID и имени пользователя можно воспользоваться утилитой PsGetSID.exe из пакета PSTools

HKEY_LOCAL_MACHINE (HKLM) - в данном разделе реестра хранятся глобальные аппаратные и программные настройки системы - записи для системных служб, драйверов, наборов управляющих параметров, общие настройки программного обеспечения, применимые ко **всем пользователям**. Это самая большая и самая важная часть реестра. Здесь сосредоточены основные параметры операционной системы, оборудования, программного обеспечения.

HKEY_USERS(HKU) - индивидуальные настройки среды для каждого пользователя системы (пользовательские профили) и профиль по умолчанию для вновь создаваемых пользователей.

HKEY_CURRENT_CONFIG (HKCC) - конфигурация для текущего аппаратного профиля. Обычно профиль один единственный, но имеется возможность создания нескольких с использованием "Панель управления" - "Система" - "Оборудование"- "Профили оборудования". На самом деле HKCC не является полноценным разделом реестра, а всего лишь ссылкой на подраздел из HKLM HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Hardware Profiles\Current

Параметры или ключи реестра имеют **имена**, представленные в обычном текстовом виде и **значения**, которые хранятся в виде стандартизованных записей определенного типа. Допустимые типы данных реестра:

REG_BINARY - двоичный параметр. Большинство сведений об аппаратных компонентах хранится в виде двоичных данных и выводится в редакторе реестра в шестнадцатеричном формате.

REG_DWORD - двойное слово. Данные представлены в виде значения, длина которого составляет 4 байта (32-разрядное целое). Этот тип данных используется для хранения параметров драйверов устройств и служб. Значение отображается в окне редактора реестра в двоичном, шестнадцатеричном или десятичном формате. Эквивалентами типа DWORD являются **DWORD_LITTLE_ENDIAN** (самый младший байт хранится в памяти в первом числе) и **REG_DWORD_BIG_ENDIAN** (самый младший байт хранится в памяти в последнем числе).

REG_QWORD - Данные, представленные в виде 64-разрядного целого. Начиная с Windows 2000, такие данные отображаются в окне редактора реестра в виде двоичного параметра.

REG_SZ - строковый параметр.

REG_EXPAND_SZ - Расширяемая строка данных. Многострочный параметр. Многострочный текст. Этот тип, как правило, имеют списки и другие записи в формате, удобном для чтения. Записи разделяются пробелами, запятыми или другими символами.

REG_RESOURCE_LIST - Двоичный параметр. Последовательность вложенных массивов. Служит для хранения списка ресурсов, которые используются драйвером устройства или управляемым им физическим устройством. Обнаруженные данные система сохраняет в разделе \ResourceMap. В окне редактора реестра эти данные отображаются в виде двоичного параметра в шестнадцатеричном формате.

REG_RESOURCE_REQUIREMENTS_LIST - двоичный параметр. Последовательность вложенных массивов. Служит для хранения списка драйверов аппаратных ресурсов, которые могут быть использованы определенным драйвером устройства или управляемым им физическим устройством. Часть этого списка система записывает в раздел \ResourceMap. Данные определяются системой. В окне редактора реестра они отображаются в виде двоичного параметра в шестнадцатеричном формате.

REG_FULL_RESOURCE_DESCRIPTOR - двоичный параметр. Последовательность вложенных массивов. Служит для хранения списка ресурсов, которые используются физическим устройством. Обнаруженные данные система сохраняет в разделе \HardwareDescription. В окне редактора реестра эти данные отображаются в виде двоичного параметра в шестнадцатеричном формате.

REG_NONE - Данные, не имеющие определенного типа. Такие данные записываются в реестр системой или приложением. В окне редактора реестра отображаются в виде двоичного параметра в шестнадцатеричном формате.

REG_LINK - Символическая ссылка в формате Юникод.

При добавлении новых параметров в реестр, необходимо задавать не только имя и значение, а также правильный тип данных.

Возможности конкретного пользователя при работе с данными реестра определяются правами его учетной записи. Далее по тексту, предполагается, если это не оговорено особо, что пользователь имеет права администратора системы.

При просмотре данных реестра в среде Windows XP, 2 подраздела с именами SAM и SECURITY, не отображаются, и доступ к ним разрешен только для локальной системной учетной записью (Local System Account), под которой обычно выполняются системные службы (system services). Обычно, учетные записи пользователей и даже администраторов, таких прав не имеют, и редактор реестра, запущенный от их имени, не отображает содержимое разделов SAM и SECURITY. Для доступа к ним нужно, чтобы regedit был запущен от имени учетной записи с правами Local System, для чего можно воспользоваться [утилитой PSEXEC](#) psexec.exe -i -s regedit.exe

Можно также воспользоваться стандартными средствами операционной системы, например, планировщиком заданий. С помощью команды **at** создаем задание на запуск regedit.exe в интерактивном режиме через 2-3 минуты от текущего времени (например- в 16час 14 мин.)

at 16:14 /interactive regedit.exe

Поскольку сам планировщик работает как системная служба, то порожденная им задача также будет выполнятся с наследуемыми правами, а ключ /interactive позволит текущему пользователю взаимодействовать с запущенным заданием, т.е. с редактором реестра, выполняющимся с правами локальной системной учетной записи.

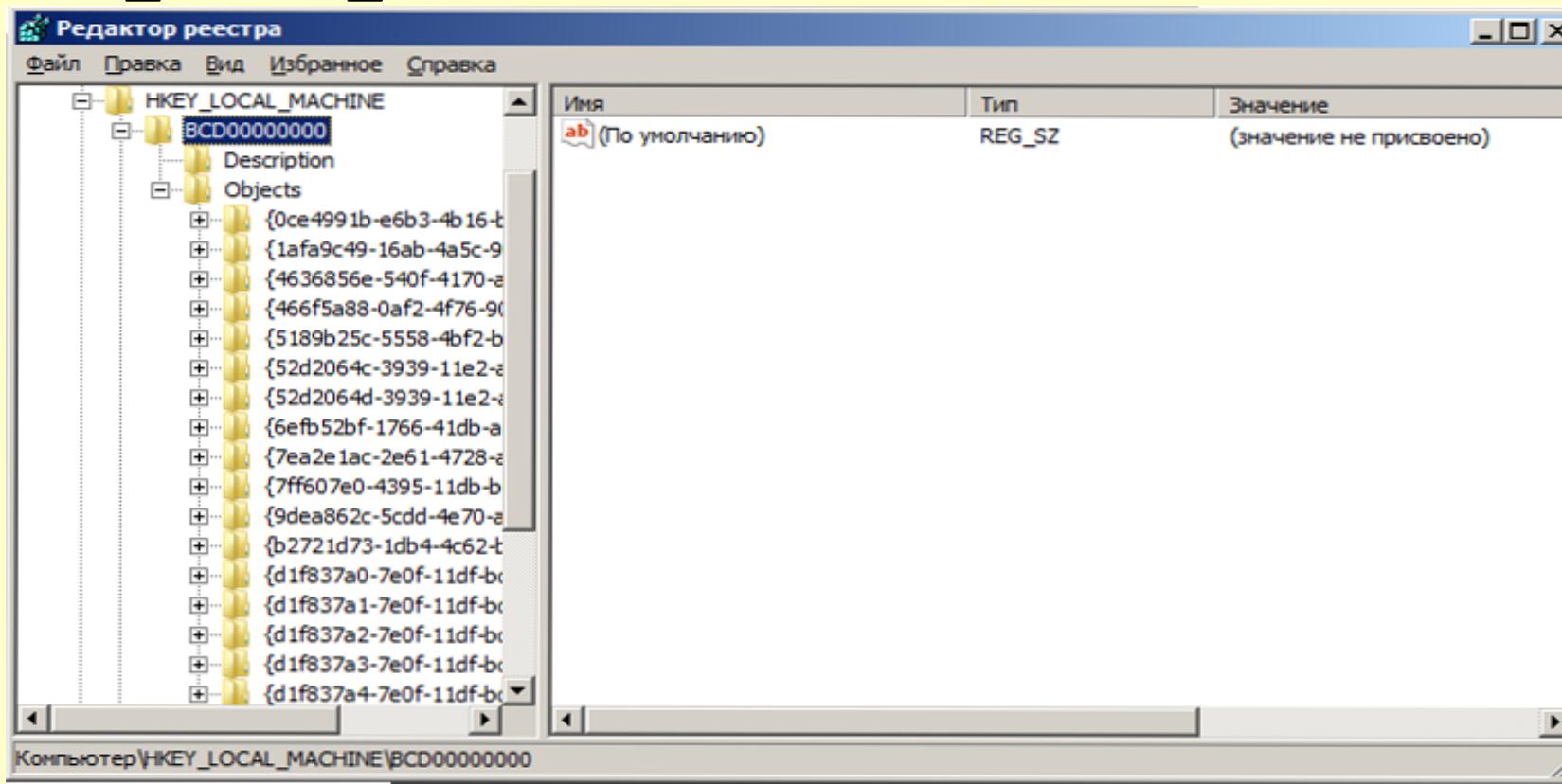
В Windows 7 разделы реестра SAM и SECURITY отображаются , однако не отображается их содержимое, для просмотра которого можно воспользоваться аналогичным приемом.

В процессе загрузки и функционирования операционной системы выполняется постоянное обращение к данным реестра, как для чтения, так и для записи. Файлы реестра постоянно изменяются, поскольку не только система, но и отдельные приложения могут использовать реестр для хранения собственных данных, параметров и настроек. Другими словами, обращение к реестру - это одна из наиболее распространенных операций. Даже если пользователь не работает за компьютером, обращения к реестру все равно выполняются системными службами, драйверами и приложениями.

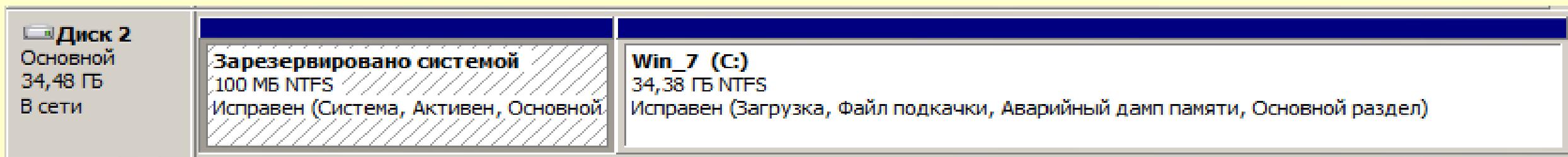
Нарушение целостности файлов реестра (нарушение структуры данных) или неверное значение отдельных критических параметров может привести к краху системы . Поэтому, прежде чем экспериментировать с реестром, позаботьтесь о возможности его сохранения и восстановления.

Особенности реестра Windows Vista и последующих версий ОС семейства Windows

Главным отличием реестра операционных систем Windows Vista / Windows 7 / Windows Server 2008 и более поздних выпусков - появление нового раздела с данными конфигурации загрузки системы **HKEY_LOCAL_MACHINE\BCD00000000**.



Этот раздел содержит объекты и элементы конфигурации, используемые новым диспетчером загрузки **BOOTMGR** пришедшим на смену традиционному загрузчику **NTLDR**. Раздел **HKEY_LOCAL_MACHINE\BCD00000000** является системным хранилищем данных конфигурации загрузки (**BCD** - Boot Configuration Data) и физически, представляет собой файл реестра с именем **bcd**, находящийся в каталоге **\BOOT** активного раздела (раздела диска с загрузочной записью и файлом диспетчера **BOOTMGR**). При стандартной установке Windows 7 на новый жесткий диск, в качестве активного раздела создается небольшой (размером около 100Мб) раздел, который содержит файлы и каталоги, необходимые для работы диспетчера загрузки. Обычно, этот раздел скрыт от пользователя, поскольку ему не присваивается буква логического диска и к его содержимому невозможно получить доступ стандартными средствами, такими, как например "Проводник" Windows (Explorer) . При просмотре с использованием Диспетчера логических дисков, данный раздел отображается под названием "Зарезервировано системой" и имеет признак "Активный"



Загрузочный сектор данного раздела (Partition Boot Sector или PBR) выполняет загрузку файла диспетчера **bootmgr**, который должен находиться в его корне. В свою очередь, диспетчер загрузки **bootmgr** для своих целей использует системное хранилище конфигурации, которое должно находиться в папке с именем **BOOT** .

Подразделы и ключи раздела HKLM\BCD00000000 имеют определенные имена, типы данных, и связи, которые обрабатываются диспетчером загрузки BOOTMGR и задают весь ход процесса дальнейшей загрузки - вид меню выбора загружаемых систем, таймаут выбора, систему, загружаемую по умолчанию, используемые устройства и приложения загрузки, и другие параметры. Иерархическая структура данных хранилища конфигурации загрузки не предназначена для редактирования с использованием редактора реестра и по умолчанию подраздел HKLM\BCD00000000 имеет разрешение только на чтение. Разрешение можно изменить с использованием контекстного меню, вызываемого правой кнопкой мыши, однако нужно учитывать то, что неквалифицированное изменение отдельных параметров данной ветви реестра может нарушить конфигурацию и системная загрузка станет невозможной. Тем не менее, экспорт данных ветви реестра HKLM\BCD00000000 иногда полезен, как дополнительная возможность анализа конфигурации загрузки в удобном для просмотра виде, а импорт - как восстановление ранее сохраненных данных BCD.

Для работы с данными конфигурации загрузки используется специальная утилита командной строки BCDEDIT.EXE , позволяющая сохранять конфигурацию, восстанавливать из ранее сохраненной копии, просматривать содержимое хранилища, редактировать отдельные объекты конфигурации и их элементы . Новый механизм загрузки операционных систем семейства Windows довольно сложен для понимания и не имеет прямого отношения к работе с реестром.

Для получения справки

REG.EXE SAVE /?

REG SAVE <раздел> <имя Файла>

<раздел> Полный путь к разделу реестра в виде: **КОРЕНЬ\Подраздел**

<КОРЕНЬ> Корневой раздел. Значения: [**HKLM | HKCU | HKCR | HKU | HKCC**].

<подраздел> Полный путь к разделу реестра в выбранном корневом разделе.

<имя Файла> Имя сохраняемого файла на диске. Если путь не указан, файл

создается вызывающим процессом в текущей папке.

Примеры:

REG SAVE HKLM\Software\MyCo\MyApp AppBkUp.hiv

Сохраняет раздел MyApp в файле AppBkUp.hiv в текущей папке

Синтаксис REG SAVE и REG RESTORE одинаков и вполне понятен из справки.

Справка самой утилиты и примеры ее использования для сохранения (REG SAVE) вполне можно использовать для сохранения любых разделов реестра, в т.ч. HKLM\software, HKLM\system и т.п. однако, если вы попробуете восстановить, например, HKLM\system, то получите сообщение об ошибке доступа, по причине занятости данного раздела реестра, а поскольку он занят всегда, восстановление с помощью REG RESTORE выполнить не удастся.

Для сохранения куста SYSTEM:

REG SAVE HKLM\SYSTEM system.hiv

Для сохранения куста SOFTWARE:

REG SAVE HKLM\SOFTWARE software.hiv

Для сохранения куста DEFAULT:

reg save HKU\.Default default.hiv

Если файл существует, то REG.EXE выдаст ошибку и завершится. В Windows 7/8/10/11 при наличии существующего файла, выдается стандартный запрос на разрешение его перезаписи.

Сохраненные файлы можно использовать для восстановления реестра с использованием ручного копированием в папку %SystemRoot%\system32\config.

Ручное копирование файлов реестра.

Этот способ возможен, если имеется копия файлов реестра, созданная на момент работоспособного состояния.

Как уже упоминалось выше, если загрузиться в другой ОС с возможностью доступа к файловой системе проблемной Windows, то с файлами из папки реестра можно делать все, что угодно.

В случае повреждения файла **system**, можно воспользоваться, например, сохраненным с помощью команды **REG SAVE** файлом **system.hiv**, скопировав его в папку реестра и переименовав в **system**.

Для Windows 7/8/10/11 – скопировать файл **system** из папки **\windows\system32\config\RegBack** в папку **\windows\system32\config**

Использование режима экспорта-импорта реестра.

Данный способ не является в полном смысле слова способом полного восстановления реестра и более подходит для случаев, когда нужно сохранить и затем восстановить определенную его часть.

Редактор реестра позволяет делать экспорт как всего реестра, так и отдельных разделов в файл с расширением *reg*. Импорт полученного при экспорте reg-файла, позволяет восстановить реестр. Щелкаете на "Реестр"-->"Экспорт (Импорт) файла реестра". Импорт также можно выполнить двойным щелчком по ярлыку reg-файла.

Вполне понятно, что наличие резервных копий реестра делает систему почти "не убиваемой", однако, нередко случается так, что при возникновении необходимости восстановления реестра актуальной копии просто нет.

Например, вирус отключил систему восстановления и удалил контрольные точки, а резервное копирование вручную просто не выполнялось. В Windows 7/8/10/11 существует задание планировщика для копирования файлов реестра, созданное при установке системы.

Что бы исключить подобную ситуацию, было бы неплохо, позаботиться об автоматическом резервирования файлов реестра без участия человека. Например, с помощью командного файла, выполняемого планировщиком или в процессе регистрации пользователя.