

Архитектура систем мобильных устройств



Лекция 1. Парадигмы программирования

Научить студентов архитектуре операционных систем с учетом их эффективного применения с программными продуктами, разрабатываемыми с целью автоматизации различных видов деятельности, с применением всех видов обеспечения современных вычислительных систем.

- Ознакомить студентов с архитектурой современных операционных систем, включая их реализации для мобильных устройств.
- Ознакомить студентов с парадигмами программирования.
- Привить практические навыки по системному программированию и администрированию.
- Освоить создание сложных клиент-серверных сетевых программных приложений

1. Нормативные акты и стандарты, регулирующие терминологию и порядок разработки и жизненного цикла программного обеспечения.
2. Виды обеспечения вычислительных систем.
3. Современные системы и парадигмы программирования.
4. Области применения ИТ специалиста.
5. Специалисты необходимые для обеспечения выполнения функциональных обязанностей.
6. Перечень дисциплин необходимых для подготовки ИТ специалистов в определенной предметной области

Нормативные акты, регулирующие терминологию программного и информационного обеспечения

ГОСТ 19781-90 - ОБЕСПЕЧЕНИЕ СИСТЕМ ОБРАБОТКИ ИНФОРМАЦИИ ПРОГРАММНОЕ.
Термины и определения.

ГОСТ 15971-90 - СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ. Термины и определения.

ГОСТ 20886-85 - ОРГАНИЗАЦИЯ ДАННЫХ В СИСТЕМАХ ОБРАБОТКИ ДАННЫХ. Термины и
определения.

ГОСТ 24402-88 - ТЕЛЕОБРАБОТКА ДАННЫХ И ВЫЧИСЛИТЕЛЬНЫЕ СЕТИ. Термины и
определения.

ГОСТ 28397-89 (ИСО 2382-15-85) - ЯЗЫКИ ПРОГРАММИРОВАНИЯ. Термины и определения.

Российские стандарты на разработку программных продуктов

6

ГОСТ 34.601 90 - Информационная технология. Автоматизированные системы.
Стадии создания.

ГОСТ 34.602 89 - Информационная технология. Техническое задание на создание
автоматизированной системы.

ГОСТ 34.201 89 - Информационная технология. Виды, комплектность и
обозначение документов при создании автоматизированных систем.

РД 50 34.698 90 - Автоматизированные системы. Требования к содержанию
документов.

ГОСТ 28195 89 - Оценка качества программных средств. Общие положения.

ГОСТ 34.603 92 - Информационная технология. Виды испытаний
автоматизированных систем.

ГОСТ 28806 90 - Качество программных средств. Термины и определения.

1. Программное обеспечение:

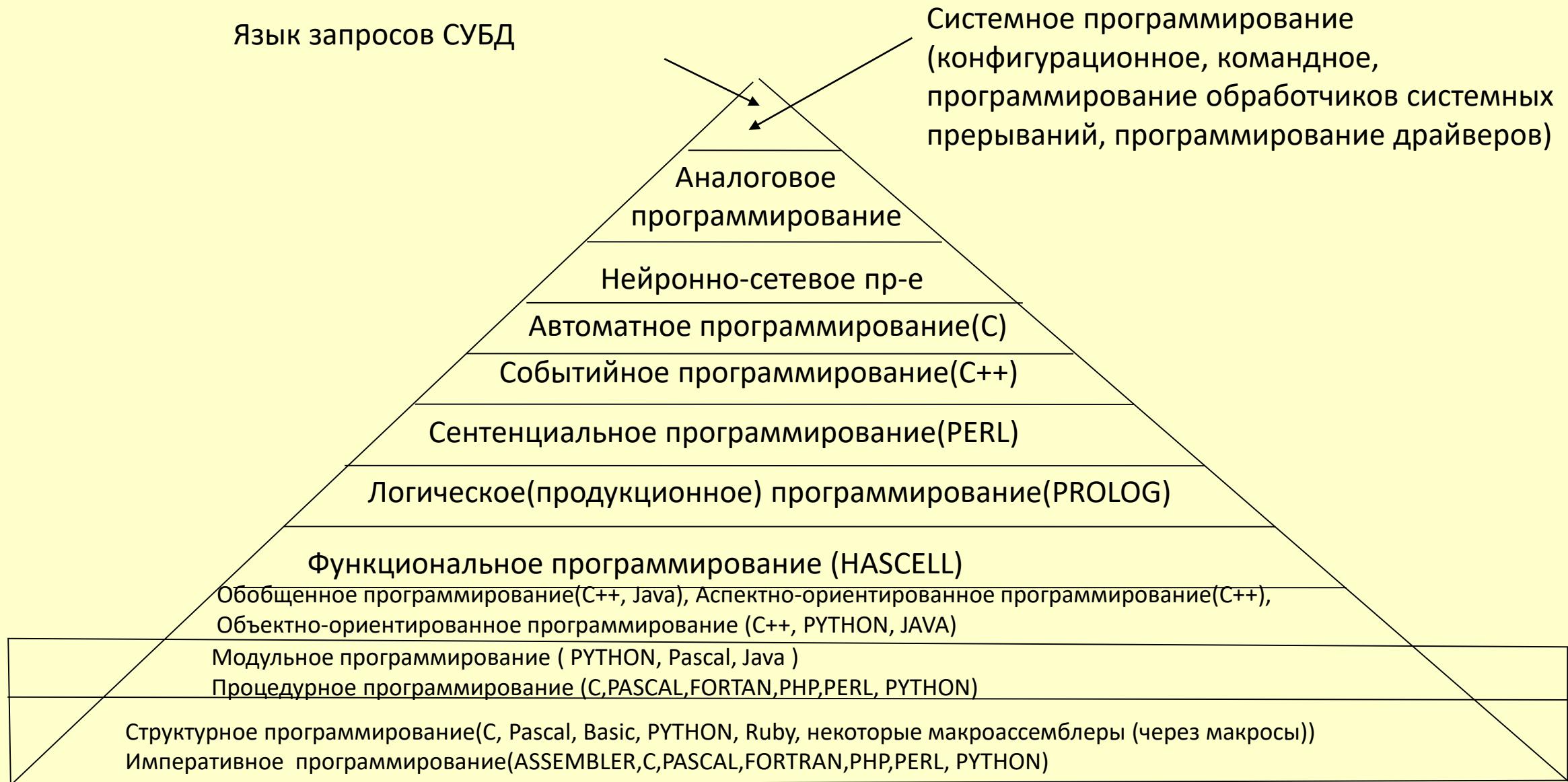
- **Общее программное обеспечение** – комплекс программ предназначенный для решения универсальных задач (операционные системы, антивирусные комплексы, браузеры, сетевые экраны, офисные приложения общего назначения, архиваторы)
- **Специальное программное обеспечение** – комплекс программ предназначенных 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

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

- 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

14

- 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.

«Современное состояние вычислительных систем и перспективы их развития»

Познакомиться с современным состоянием вычислительных систем и перспективами их дальнейшего развития

1. Современное состояние вычислительных систем.
2. Основные направления развития вычислительных систем

Основная

1. Симонович С. В. Информатика. Базовый курс. Учебник для вузов. - СПб.: Питер, 2015. - 640 с. (ЭБС)
2. Вербицкий А.С., Волков А.Г., Кукушкин И.А. Информатика: Электронное учебное пособие. – Балашиха: ВА РВСН, 2016г. (ЭБС)
3. Макарова Н.В., Волков В.Б. Информатика: Учебник для вузов. – СПб.: Питер, 2015. – 576 с. (ЭБС)

Дополнительная

1. Уткин В.Б. Информационное обеспечение систем управления: Учебник.- М.: РВСН, 2012. – 423 с. Стр. 378 – 390
2. Базы знаний интеллектуальных систем/ Т.А. Гаврилова, В.Ф. Хорошевский – СПб.: Питер, 2015. – 384 с. Стр. 39 – 59

1 вопрос

Современное состояние
вычислительных систем.

Виды обеспечения вычислительных систем

- Техническое обеспечение
- Программное обеспечение
- Информационное обеспечение
- Организационное обеспечение

Современное техническое обеспечение вычислительных систем

Техническое обеспечение вычислительных систем - совокупность аппаратных средств предназначенных для реализации программного и информационного обеспечения:

- Рабочие станции на основе автоматизированных рабочих мест, тонкие клиенты(х-терминалы);
- Серверы,NAS-накопители, мейнфреймы, кластерные структуры,DATA-центры;
- Сетевая инфраструктура - коммутаторы 3 поколения(имеют внутренние таблицы соответствия MAC адресов и портов ввода-вывода), коммутаторы 4 поколения (программируемые), маршрутизаторы, кабельная продукция (коаксиальный кабель, витая пара, оптоволоконный кабель), аппаратура передачи данных (в том числе беспроводная), сетевые аксессуары (специальные шкафы, сетевые розетки),система обеспечения безопасности обработки информации(криптомуартизаторы, аппаратные сетевые экраны, аппаратура разграничения доступа), системы охлаждения, система охранной сигнализации и противопожарной безопасности;
- Аппаратные комплектующие, входящие в состав выше указанных систем – микропроцессоры(CPU,APU), HDD и SSD накопители, блоки оперативной памяти, материнские платы с набором микросхем обеспечения, блоки питания

Современные центральные микропроцессоры (универсальные CPU и APU - зарубежные образцы)

топовые desktop варианты фирм AMD и Intel (75000 рублей)

Процессор	AMD Ryzen 9 5950X	Core i9-10980XE
Название ядра	Vermeer (Zen3)	Cascade Lake
Технология производства	7 нм	14 нм
Частота ядра, ГГц	3,4/4,9	3/4,8
Количество ядер/потоков	16/32	18/36
Кэш L1 (сумм.), I/D, КБ	64КБ*16	18*64КБ
Кэш L2,	16*512КБ	18*1МБ
Кэш L3, МБ	64	24.75
Оперативная память	2×DDR4-3200	4×DDR4-2933
TDP, Вт	105	165
Количество линий PCIe 3.0	20	48

Производительность данных процессоров практически равная и очень избыточна для решения большинства используемых задач, в том числе задач искусственного интеллекта (распознавание объектов, переводы документов и т.д.)

Современные центральные микропроцессоры (универсальные CPU и APU - зарубежные образцы)

бюджетные desktop варианты фирм AMD(10000 рублей) и Intel(11000 рублей)

Процессор	AMD Ryzen 3 3200G	Intel Core i3-8100
Название ядра	Raven Ridge	Coffee Lake
Технология производства	12 нм	14 нм
Частота ядра, ГГц	3,6/4	3,6
Количество ядер/потоков	4/4	4/4
Кэш L1 (сумм.), I/D, КБ	384	128/128
Кэш L2, КБ	2000	4×256
Кэш L3, МБ	4	6
Оперативная память	2×DDR4-2933	2×DDR4-2400
TDP, Вт	65	65
Graphics	Vega 8	

Производительность данных процессоров практически равная и достаточна для решения большинства используемых задач, в том числе задач искусственного интеллекта (распознавание объектов, переводы документов и т.д.)

Современные центральные микропроцессоры (универсальные CPU и APU – отечественные образцы)

Эльбрус-8С фирмы МЦСТ (серийный выпуск с 2016 года)
для серверов и рабочих станций

Технические характеристики

Архитектура Эльбрус, версия 4

Масштабируемость 8 ядер в процессоре

4 процессора в модуле (16 Гбайт/с попарные связи)

2 модуля в машине

Тактовая частота 1300 МГц (1891ВМ10АЯ)

1000 МГц (1891ВМ10БЯ)

Пиковая производительность 25 операций в такт в каждом ядре (8 цел., 12 веществ.)

250 GFLOPS одинарной точности, **125 GFLOPS** двойной точности

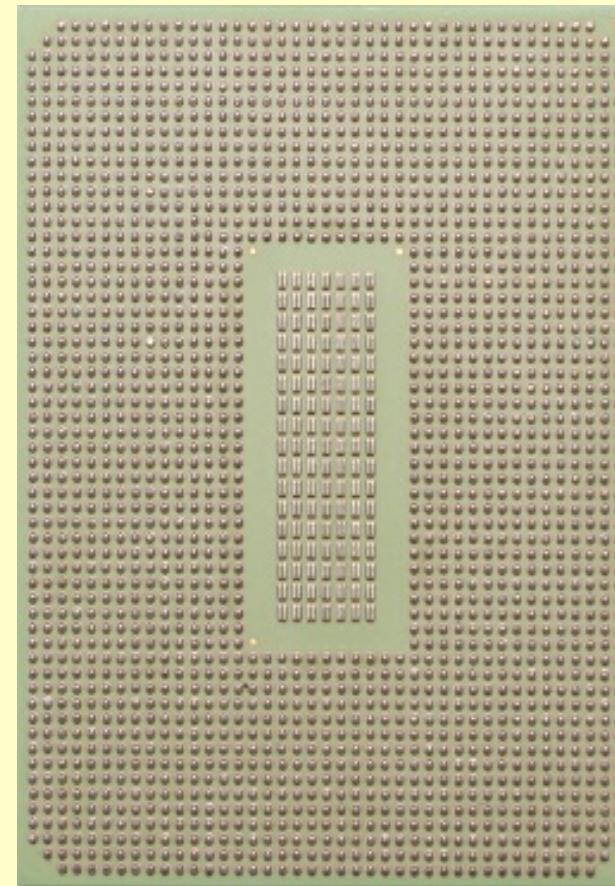
Кэш-память L1: 64 Кбайт данные + 128 Кбайт команды в каждом ядре

L2: 512 Кбайт в каждом ядре, 4 Мбайт суммарно

L3: 16 Мбайт в процессоре

- Оперативная память 4 канала DDR3-1600 registered ECC, до 51,2 Гбайт/с
- 64 Гбайт на процессор
- 1 Тбайт адресное пространство машины
- Периферия 1 канал ввода-вывода, до 16 Гбайт/с
- совместимый контроллер — КПИ-2
- Технологические параметры
- Топология 2,73 млрд. транзисторов
- 28 нм техпроцесс, 321 мм² площадь кристалла
- Корпус 59,5×43,0×4,6 мм, 32,0 г
- 2028 контактов FCBGA
- Электропитание 0,9 В, 1,0 В, 1,15 В, 1,5 В, 1,8 В
- 80 Вт (1891ВМ10АЯ) Условия эксплуатации -60...+85 °C
- 60 Вт (1891ВМ10БЯ) -40...+90 °C

Микросхема центрального процессора 1891ВМ10Я — высокопроизводительный вычислитель серверного класса. Содержит 8 ядер архитектуры «Эльбрус» 4-го поколения с тактовой частотой до 1300 МГц. Позволяет строить многопроцессорные серверы и рабочие станции, а также бортовые вычислители, требовательные к скорости обработки и передачи информации.



Процессор Эльбрус-16С

Содержит 16 вычислительных ядер.

Общая производительность 1,5 ТФлопс одинарной точности и 750 ГФлопс двойной точности.

8 каналов памяти DDR4-3200 ECC.

Встроенные контроллеры Ethernet 10 и 2,5 Гбит/с.

32 линии PCI-Express 3.0.

4 канала SATA 3.0.

Он поддерживает объединение в многопроцессорные системы до 4 процессоров с общим объёмом оперативной памяти до 16 Тбайт. Общее число транзисторов в процессоре составит 12 млрд.

По многим пунктам этот процессор должен стать первым в России:

Первый процессор по технологии 16 нм, спроектированный в России и основанный на российских технологиях.

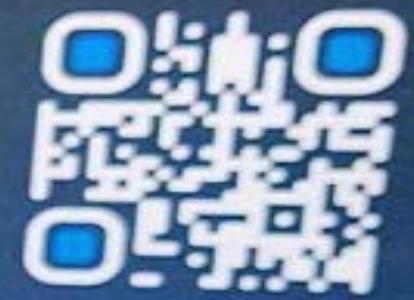
Первый универсальный микропроцессор Эльбрус с аппаратной поддержкой виртуализации.

Первый микропроцессор Эльбрус, штатно работающий на частоте 2 ГГц.

Первый российский универсальный процессор с реализацией 8 каналов DDR4-3200 ECC.



Эльбрус



Эльбрус-16С
0318M038
Р04705-0001
2014

ЭЛЬБРУС-16С
ИНЖЕНЕРНЫЙ
ОБРАЗЕЦ

эльбрус

Современные центральные микропроцессоры (универсальные CPU и APU – отечественные образцы) Комдив64 (1907ВМ028) фирмы ФГУ ФНЦ НИИСИ РАН (серийный выпуск с 2016 года), стойкая к воздействию радиационного облучения микросхема

64-разрядная система на кристалле для построения высокопроизводительных вычислительных комплексов с повышенной радиационной стойкостью и стойкостью к воздействию частиц космического пространства.

Состав интерфейсов и устройств:

- контроллер динамической памяти;
- контроллер последовательного RapidIO 4X (или 2 канала 1X);
- контроллер интерфейса PCI;
- контроллер прерываний;
- блок таймеров;
- контроллер ППЗУ;
- контроллер Ethernet;
- контроллер I2C.

Основные технические характеристики:

- 64-разрядная RISC архитектура КОМДИВ64;
- поддержка 32-разрядного режима выполнения инструкций и режима адресации;
- сопроцессор вещественной арифметики с форматами вещественных чисел одинарной (32 разряда) и двойной (64 разряда) точности, а также "пара вещественных чисел одинарной точности";
- специализированный сопроцессор комплексной арифметики с отдельным регистровым файлом на 64 64-разрядных комплексных регистра;
- трансляция 32-разрядных и 64-разрядных виртуальных адресов в 36-разрядные физические;
- ассоциативный буфер трансляции виртуальных адресов (jTLB) на 64 адреса (128 страниц);
- раздельные кэши первого уровня (инструкций и данных) по 16 Кбайт каждый (4 секции);
- кэш-память 2-го уровня размером 256 Кбайт с возможностью работы в режиме накристальной памяти;
- встроенный контроллер DMA для пересылок между внешней и накристальной памятью;
- 7-ступенчатый суперскалярный конвейер с предвыборкой, переупорядочиванием и возможностью выполнения двух команд за такт;
- считывание до четырех команд за один такт;
- статическое предсказание переходов и спекулятивное выполнение инструкций;
- потребляемая мощность на частоте 150 МГц не более 5,5 Вт, 66 МГц - не более 2 Вт;
- корпус керамический матричный 675 выводов.

Современные центральные микропроцессоры (универсальные CPU и APU – отечественные образцы)

Байкал-М фирмы «Байкал электроникс» (выпуск с 2017 года) для рабочих станций и серверов

- Основные характеристики процессора Байкал-М
- 8 ядер ARM Cortex-A57 (разрядность 64 бит).
- Рабочая частота до 2 ГГц.
- Аппаратная поддержка виртуализации и технологии Trust Zone на уровне всей СнК.
- Интерфейс с оперативной памятью – два 64-битных канала DDR3/DDR4-2133 с поддержкой ECC
- Кэш-память – 4 МБ (L2) + 8 МБ (L3).
- Восьмиядерный графический сопроцессор Mali-T628.
- Видеотракт, обеспечивающий поддержку HDMI, LVDS
- Аппаратное декодирование видео
- Встроенный контроллер PCI Express поддерживает 16 линий PCIe Gen. 3.
- Два контроллера 10-гигабитной сети Ethernet, два контроллера гигабитной сети Ethernet. Контроллеры поддерживают виртуальные сети VLAN и приоритетацию трафика.
- Два контроллера SATA 6G, обеспечивающих скорость обмена данными до 6 Гбит/с каждый.
- 2 канала USB v.3.0 и 4 канала USB v.2.0.
- Поддержка режима доверенной загрузки.
- Аппаратные ускорители, поддерживающие ГОСТ 28147-89, ГОСТ Р 34.11-2012.
- Энергопотребление – не более 30 Вт.



Области применения Байкал-М

- моноблок, автоматизированное рабочее место, графическая рабочая станция;
- домашний (офисный) медиа-центр;
- сервер и терминал видеоконференций;
- микросервер;
- NAS уровня небольшого предприятия;
- маршрутизатор / брандмауэр.

Широкое распространение архитектуры ARMv8 (AArch64) позволяет использовать огромное количество готового прикладного и системного программного обеспечения.

Поддерживаются операционные системы Linux и Android, в том числе на уровне бинарных дистрибутивов и пакетов.

Доступны драйверы многочисленных устройств, подключаемых к шинам PCIe и USB.

В состав поставляемого «Байкал Электроникс» комплекта программного обеспечения входит ядро Linux в исходных текстах и скомпилированном виде, а также драйверы для встроенных в Baikal-M контроллеров.

Современные центральные микропроцессоры (универсальные CPU и APU – отечественные образцы)

Байкал-T1 фирмы «Байкал электроникс» (серийный выпуск с 2016 года)

Технические характеристики

- 2 ядра P5600 MIPS 32 r5, максимальная частота до 1,2 ГГц
- Кэш L2 1 Мбайт
- Контроллер памяти DDR3-1600
- Энергопотребление не более 5 Вт
- Технологический процесс 28 нм
- Интегрированные интерфейсы:
 - 1 порт 10 Gb Ethernet
 - 2 порта 1 Gb Ethernet
 - контроллер PCIe Gen.3
 - 2 порта SATA 3.0
 - USB 2.0

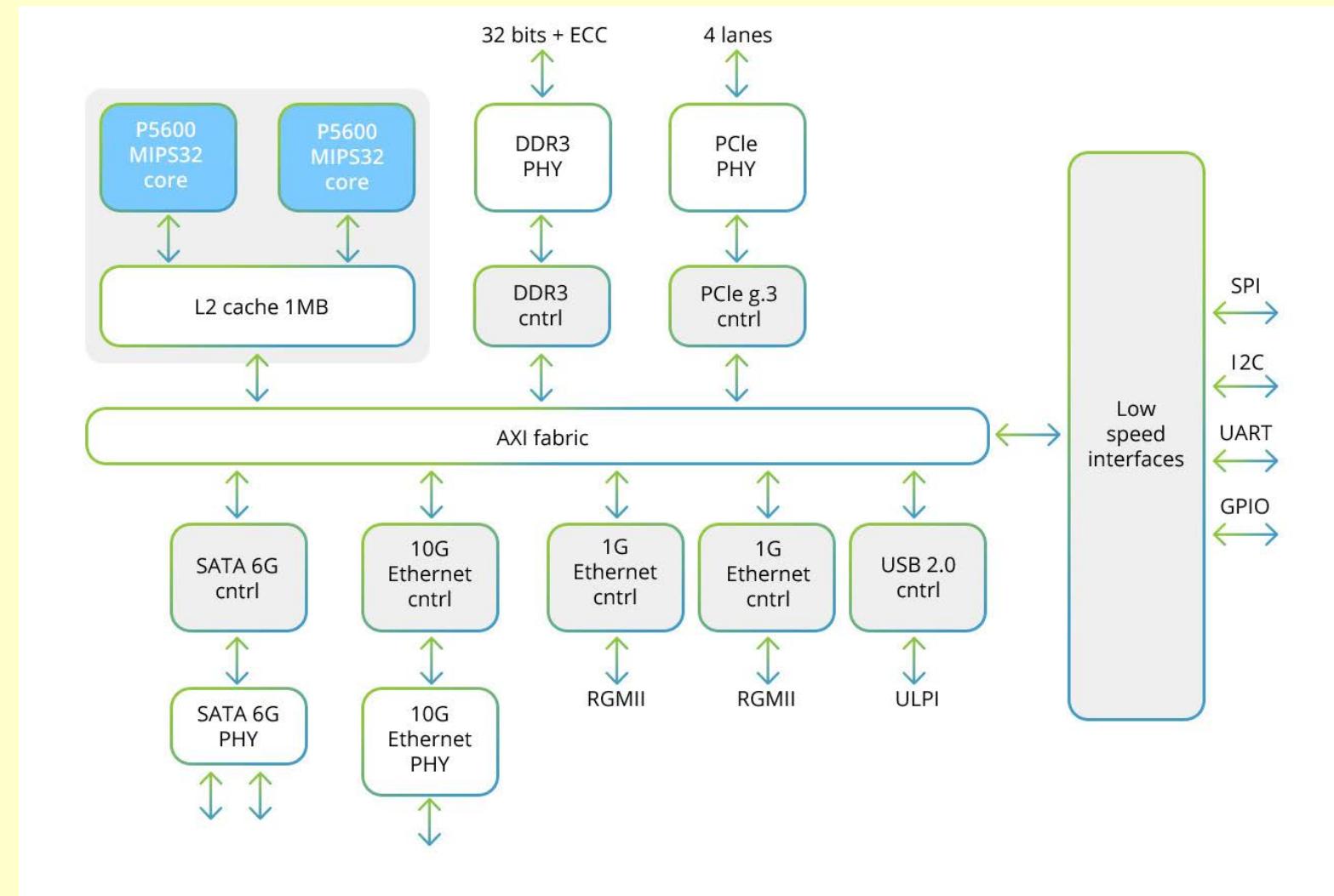
Сфера применения

Коммуникационная инфраструктура

Офисные рабочие места

Сетевые накопители данных

Промышленная автоматизация и управление зданиями



Современные центральные микропроцессоры (универсальные CPU и APU – отечественные образцы)

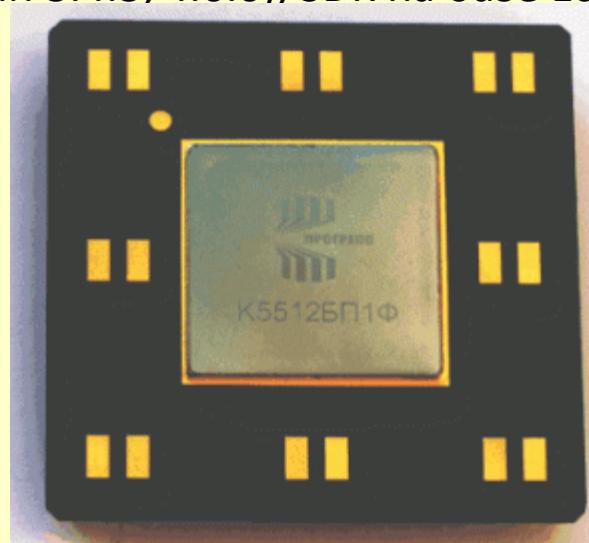
KVARC компании **KM211**(производится в данное время)

- Микропроцессор **KVARC**, собственная разработка компании **KM211**, представляет собой 32-разрядный **RISC** микропроцессор общего назначения для встраиваемых применений. В основе лежит полностью российская, лицензионно чистая разработка на высокопроизводительном ядре с малым количеством логических вентилей и низким энергопотреблением.
- Классическая архитектура
- RISC, гарвардская архитектура
- 32-разрядные операнды
- 4Гб адресное пространство
- 5-ти стадийный конвейер
- Большинство инструкций выполняются за 1 процессорный цикл
- Статическое предсказание переходов
- Диспетчер памяти (MMU) с аппаратной подкачкой страниц
- 32-разрядная системная шина

Быстродействие-энергопотребление на уровне процессоров ARM9/ARM11

1450 МГц по технологии TSMC 28HPC

- 1.37 DMIPS/МГц
- 0,7 MWIPS/МГц
- Высокая энергоэффективность 8,5 DMIPS/мВт (для сравнения ARM946E-E: 1,19 DMIPS/МГц и 8,6 DMIPS/мВт)
- Умножение с накоплением 16×16 , 2 такта
- DSP расширение набора команд
- 32/16-разрядный набор команд, высокая плотность кода
- FPU SP/DP опциональный модуль плавающей арифметики с одинарной и двойной точностью
- Опциональный аппаратный кодек (видео MPEG2/MPEG4 кодирование-декодирование, аудио MP3)
- Спящий режим с низким потреблением
- Порттированный FreeRTOS, Linux 2.6.34; Linux 3.4, Android
- Си-компилятор GNU (GCC версии 3.4.3, 4.6.0), SDK на базе Eclipse
- JTAG, отладчик GDB



Применение

Портативные устройства:

- персональный КПК сотрудника МВД
- управление навигационными системами
- считыватель для биометрических загранпаспортов, водительских прав, банковских карт

Цифровая обработка сигналов, мультимедиа:

- цифровое ТВ, set-top-box
- управление камерами наружного наблюдения с высоким разрешением

Зашита каналов передачи данных:

- крипто-модуль для радиостанций сотрудников спецслужб
- сетевой роутер с поддержкой российских протоколов защиты информации

USB идентификатор пользователя РС

Промышленные системы управления, системы управления сигналами

Автомобильная промышленность:

- автосигнализация
- чёрный ящик для автомобиля видеорегистратор (нужен при страховых разбирательствах)

Военное применение, авионика

Медицинская аппаратура

Полностью российская передовая разработка (энергоэффективность на уровне лучших мировых микропроцессорных ядер)

Полный комплект конструкторской документации и сопровождающей информации в соответствии с российскими требованиями и стандартами

Возможность изготовления по стойким технологиям, военная приёмка

Современные центральные микропроцессоры (универсальные CPU и APU – отечественные образцы)

МультиКлет R1 компании Мультиклет (выпуск с марта 2015 года)

- Микропроцессор **MCp0411100101** имеет в своем составе мультиклеточное процессорное ядро – первое процессорное ядро с принципиально новой (пост-неймановской) мультиклеточной архитектурой российской разработки. Мультиклеточный процессор предназначен для решения широкого круга задач управления и цифровой обработки сигналов в приложениях, требующих минимального энергопотребления и высокой производительности.
- Данный мультиклеточный процессор состоит из 4 клеток (когерентных процессорных блоков), объединенных интеллектуальной коммутационной средой.

Краткие основы архитектуры

Процессор R1, как и первый представитель мультиклеточной архитектуры, состоит из четырех клеток. В архитектуру заложены следующие основные принципы:

клетки независимы и идентичны

- никто и ничто не управляет клетками, нет центрального блока управления
- клетки могут быть объединены в любую конфигурацию в любом количестве
- прямая связность инструкций по данным (в качестве аргумента инструкции напрямую указывается инструкция, результат которой нам необходим)
- одна и та же программа может быть выполнена на любом количестве клеток
- работаем тогда, когда есть работа (т.е. когда данных нет, инструкции, от них зависящие, не выполняются)
- все команды готовые к выполнению, выполняются одновременно (в каждом такте могут быть выполнены по 1 команде из каждого блока ALU_INTEGER, ALU_FLOAT, DMS и так в каждой клетке)
- динамическое распределение ресурсов



Современные микропроцессоры (специального назначения – отечественные образцы)

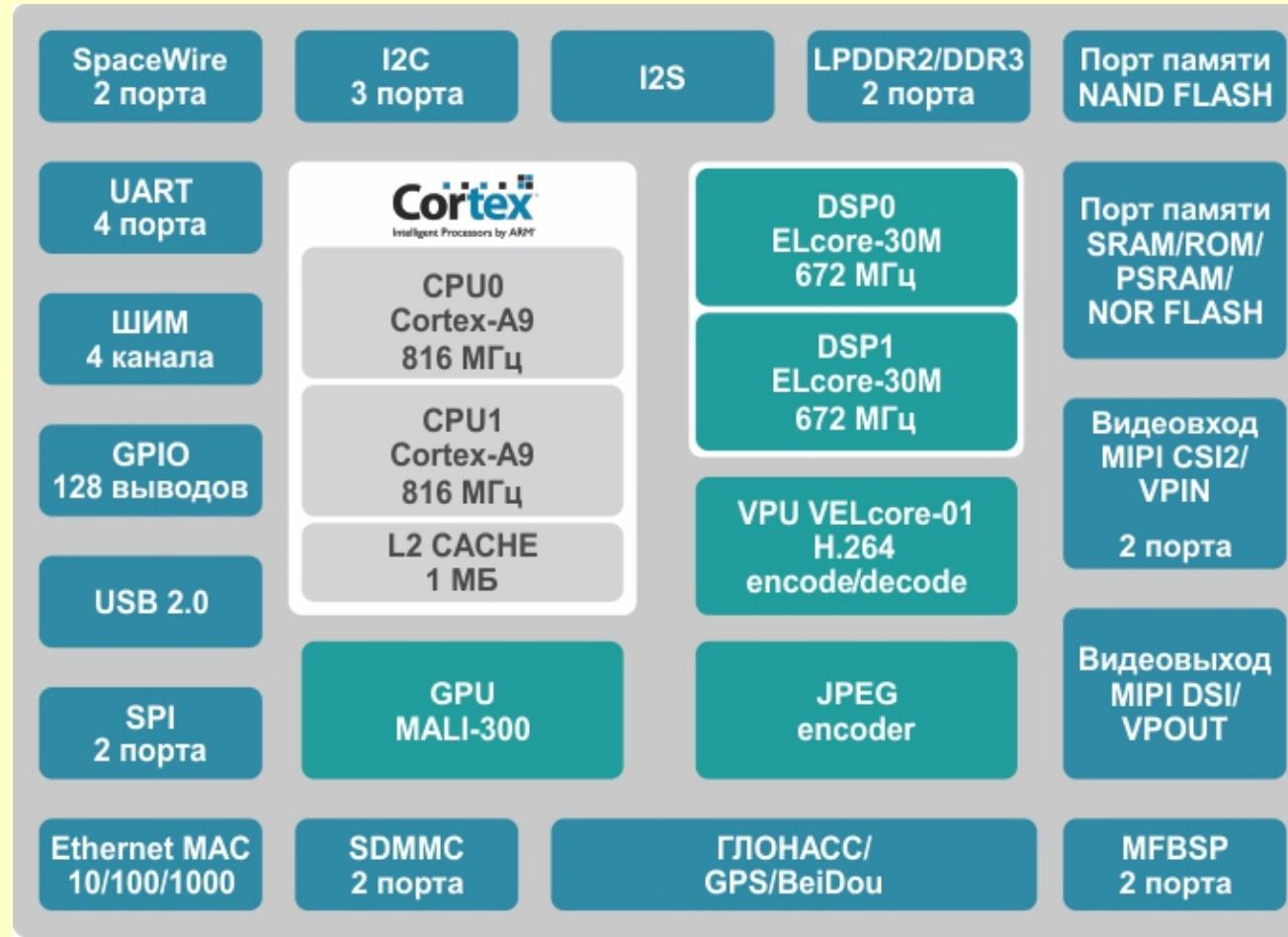
NeuroMatrix (1879ВМ4 (NM6405)) компании НТЦ “Модуль” (Выпуск с 2009 года)

1879ВМ4 (NM6405) – высокопроизводительный процессор цифровой обработки сигналов с векторно-конвейерной VLIW/SIMD архитектурой на базе запатентованного 64-разрядного процессорного ядра **NeuroMatrix**. Технология изготовления – 250 нм КМОП, тактовая частота 150 МГц. Благодаря ряду аппаратных особенностей микропроцессоры этой серии могут быть использованы не только в качестве специализированных процессоров цифровой обработки сигналов, но и для **создания нейронных сетей**. Позволяет очень эффективно решать задачи искусственного интеллекта.

Современные микропроцессоры (специального назначения – отечественные образцы)

1892ВМ14Я компании ГУП НПЦ “ЭЛВИС”(выпускается в данное время)

- Малопотребляющий многоядерный сигнальный микропроцессор нового поколения 1892ВМ14Я для связных, навигационных, мультимедийных, встраиваемых, мобильных приложений, например: планшетов, интеллектуальных видеокамер, телефонов.
- Представляет собой высокопроизводительную микропроцессорную систему на кристалле, включающую два DSP ядра ELcore-30M, два CPU ARM Cortex-A9, кодек H.264, графический 3D акселератор Mali-300, навигационный коррелятор ГЛОНАСС/GPS/Beidou и встроенные порты ввода/вывода.
- Микросхема изготовлена по технологии КМОП с минимальными топологическими размерами элементов 40 нм.



CPU микросхемы поддерживает ядро ОС **Linux 2.6.19** или ОС жесткого реального времени **QNX 6.3 (Neutrino)**.

Современные микропроцессоры (специального назначения – отечественные образцы)

Южный мост 1991ВГ2Я компании АО “МЦСТ”(выпускается в данное время)

Микросхема южного моста 1991ВГ2Я – контроллер ввода-вывода 2-го поколения собственной разработки МЦСТ, обеспечивающий скорость обмена данными с центральным процессором до 16 Гбайт/с.



Технические характеристики

Тактовая частота	500 МГц в работе, 250 МГц в простое (1991ВГ2АЯ) 450 МГц в работе, 233 МГц в простое (1991ВГ2БЯ)
Канал к процессору	16 Гбайт/с (8 на приём + 8 на передачу)
Контроллеры PCI	1 шина PCI-E 2.0 на 16 линий (1×16 или 2×8) 1 шина PCI-E 2.0 на 4 линии (1×4 или 2×2 или 4×1) 1 шина PCI 2.0, 32 бита, 33/66 МГц, до 7 устройств bus master

Контроллеры Ethernet	3 порта 1000Base-TX (802.3ab) с поддержкой PTP v2
Контроллеры ATA	8 портов SATA 3.0 (6 Гбит/с) 1 порт на 2 устройства IDE ATA (100 Мбайт/с)
Контроллеры USB	8 портов USB 2.0 (480 Мбит/с)
Контроллеры HD Audio	5.1-канальный звук, 8–32 бита, 6–192 кГц
Контроллеры LPC	2 порта RS-232/485 (115,2 кбит/с) 1 порт IEEE 1284 с поддержкой DMA (2,5 Мбайт/с) 4 канала I ² C, 1 МГц 1 шина SPI на 4 устройства, 50 МГц 32 входа-выхода GPIO
Обработка событий	16+24 входа на контроллере внешних прерываний 1 системный таймер 1 сторожевой таймер, 32 бита, 10 МГц 3 линии приёма-выдачи прерываний привязки времени

Современное программное обеспечение вычислительных систем

Общее программное обеспечение(для рабочих станций)

- Операционные системы
- Сетевое программное обеспечение
- Офисное программное обеспечение
- Программный комплекс для обработки растровой графики
- Программный комплекс для обработки векторной графики
- Программный комплекс для обработки аудио информации
- Программный комплекс для обработки видео информации
- Программный комплекс для издательских систем

Современные операционные системы

Семейство закрытых, не свободных операционных систем на базе платформы MS Windows:

- MS Windows-7,8,8.1,10-версии для рабочих станций и автономных ПЭВМ;
- MS Windows Server 2003,2008,2012,2016,2019-версии для серверов, мейнфреймов и кластерных структур;

MS Windows-7 - операционная система сертифицированная для обработки секретной и совершенно секретной информации.

Семейство открытых, свободных операционных систем на базе платформы Linux:

- Linux Ubuntu 20.04 LTS, Linux Mint Mate 20.x x64, Linux Mint Cinemon x, Astra Linux Special Edition v.1.3-1.6, MCBC-5.0 - версии для рабочих станций и автономных ПЭВМ;
- Linux Lubuntu 16.04 LTS – версия для устаревших, маломощных компьютеров;
- Astra Linux Special Edition v.1.3-6, Linux Ubuntu Server 16.04 LTS, FreeBCD 9.x, MCBC-5.0 Server - версии для серверов, мейнфреймов и кластерных структур;

Astra Linux Special Edition v.1.3-1.6 - операционная система сертифицированная для обработки секретной и совершенно секретной информации.

Сетевое программное обеспечение

- Браузер(Mozilla) – клиентский программный комплекс предназначенный для работы с WEB серверами компьютерной сети;
- FTP клиент(Filezilla) - клиентский программный комплекс предназначенный для работы с FTP серверами компьютерной сети;
- Почтовый клиент (Thunderbird) - клиентский программный комплекс предназначенный для работы с почтовыми серверами компьютерной сети;

Офисное программное обеспечение

Для платформы MS Windows:

- MS Office 2003,2007,2010,2013,2016
- LibreOffice 6.x
- WPS Office 10.x
- Adobe Acrobat Reader
- FineReader 14

Для платформы Linux:

- LibreOffice 6.x
- WPS Office 10.x
- Adobe Acrobat Reader
- Утилита - просмотр документов
- Утилита – простое сканирование

Программный комплекс для обработки векторной графики

Для платформы MS Windows:

- Программный комплекс CorelDraw-8
- Программный комплекс LibreDraw-6.0
- Программный комплекс Lnkscape-0.91

Для платформы Linux:

- Программный комплекс LibreDraw-6.0
- Программный комплекс Lnkscape-0.91

Программный комплекс для обработки растровой графики

Для платформы MS Windows:

- Программный комплекс Photoshop CC 2017
- Программный комплекс Gimp
- Программный комплекс Paint

Для платформы Linux:

- Программный комплекс Gimp

Программный комплекс для обработки аудио информации

Для платформы MS Windows:

- Sound Forge Pro 11.0 – аудио редактор
- VLC media player- проигрыватель
- The KMPlayer - проигрыватель
- Total Audio Converter – аудио конвертор

Для платформы Linux:

- VLC media player- проигрыватель
- Программный комплекс Audaciti – аудио редактор

Программный комплекс для обработки видео информации

Для платформы MS Windows:

- Программный комплекс “Видеомастер” – видео редактор;
- Программный комплекс “Видеомонтаж” – видео редактор;
- Программный комплекс Camtasia – запись происходящего на экране;

Для платформы Linux:

- Программный комплекс Openshot Video Editor – видео редактор;
- Программный комплекс Kdenlive – видео редактор;

Программный комплекс для издательских систем

Для платформы MS Windows:

- Программный комплекс QuarkXPress – издательская система
- Программный комплекс PageMaker – издательская система

Для платформы Linux:

- Программный комплекс Scribus – издательская система

Современное информационное обеспечение

Корпоративные системы управления базами данных(СУБД)

Для платформы MS Windows:

- Oracle 12 и выше
- MS SQL 2008,2012,2014
- MySQL
- PostGreSQL
- MariaDB

Для платформы Linux:

- MySQL
- PostGreSQL
- MariaDB

2 вопрос

Перспективы развития
вычислительных систем.

Краткосрочные перспективы развития программного и информационного обеспечения в отечественных вычислительных системах

- Быстрый переход на открытое и свободное программное обеспечение(выполнение программы импортозамещения)
- Оптимизация кода программ по направлению минимизации использования медленной внешней памяти
- Базы данных будут размещаться в быстрой оперативной памяти большого объема
- Активное развитие не реляционных баз данных и соответствующих СУБД

Среднесрочные перспективы развития вычислительной техники

- Создание эффективных структурных квантовых элементов для вычислительных систем
- Создание квантового компьютера на существующей архитектуре (замена кремневых элементов позволит увеличить производительность вычислительных систем в 100 раз)
- Создание квантового компьютера с новой архитектурой (производительность вычислительных систем увеличится в миллиарды раз)

Лекция
Жизненный цикл
программного обеспечения
(SWEVOK. Основные области знаний)

Ядро знаний SWEBOK

SWEBOK (software engineering body of knowledge)

основной научно-технический документ по программной инженерии, отображающий знания и накопленный опыт специалистов по программной инженерии.

Ядру знаний SWEBOK соответствует стандарт ISO/IEC TR 19759:2005.

Версии:

- ▶ 2004 г. (SWEBOK V2) – десять областей знаний (5 основных и 5 областей управления);
- ▶ 2013 г. (SWEBOK V3) – пятнадцать областей (+ теоретические основы ПИ, экономика и описание профессиональных навыков по ПИ).

СУБВОК

Требования

Проектирование

Конструирование

Тестирование

Поддержка и эксплуатация

Основы требований

Процесс

Извлечение требований

Анализ требований

Спецификация требований

Утверждение требований

Практические соображения

Основы проектирования

Ключевые вопросы проектирования

Структура и архитектура

Анализ качества и оценка дизайна

Нотации дизайна

Стратегии и методы проектирования

Основы конструирования

Управление конструированием

Практические соображения

Основы тестирования

Уровни тестирования

Техники тестирования

Метрики, связанные с тестированием

Процесс тестирования

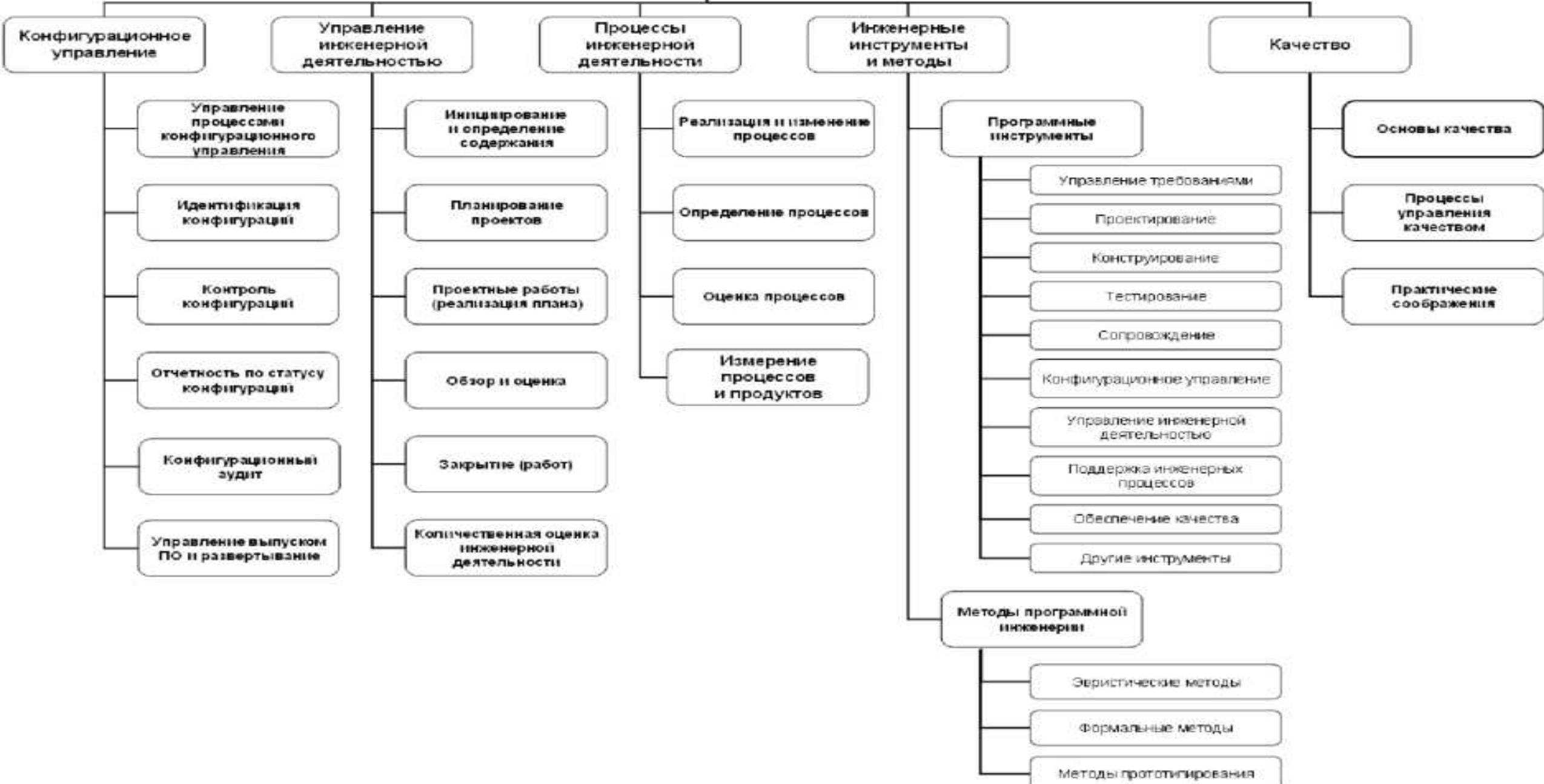
Основы поддержки и эксплуатации

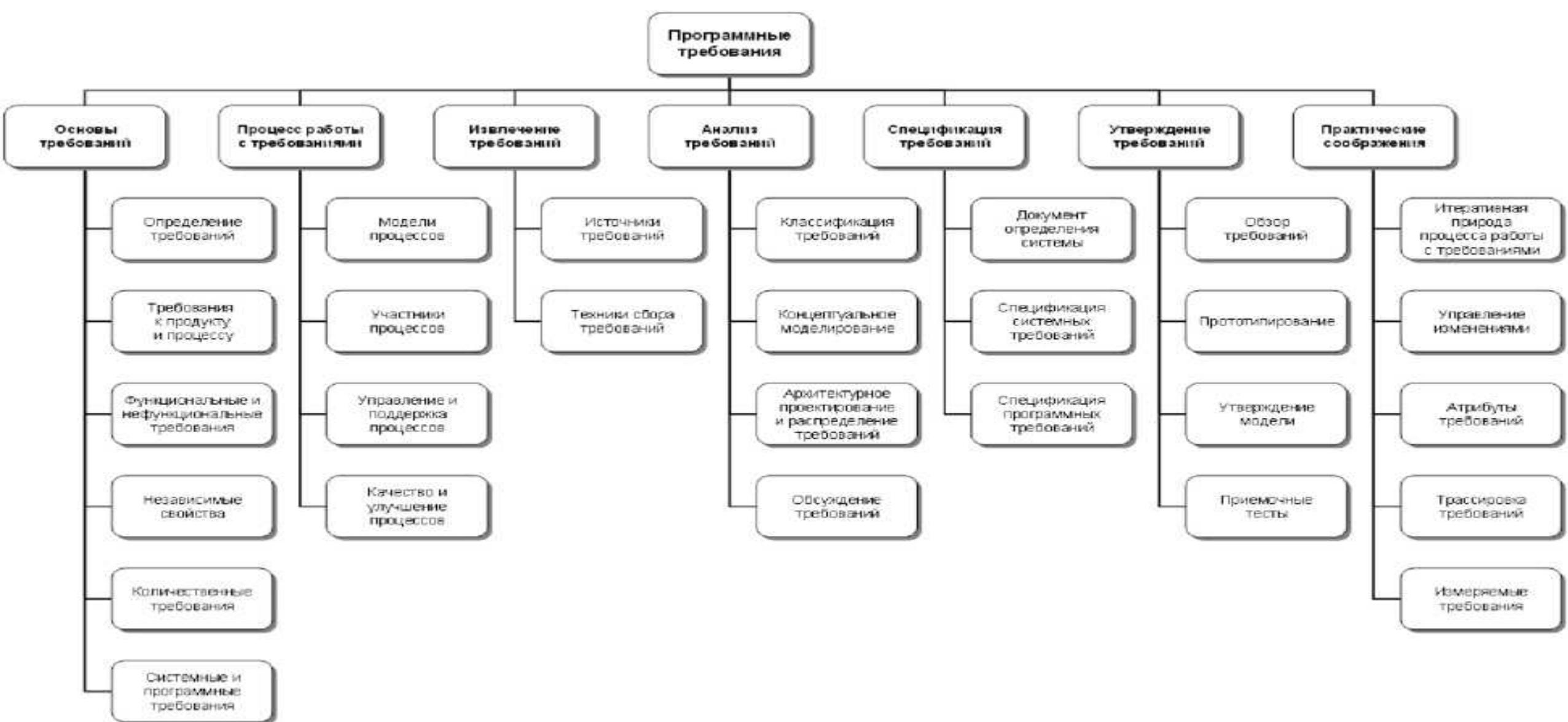
Ключевые вопросы поддержки и эксплуатации

Процесс

Техники

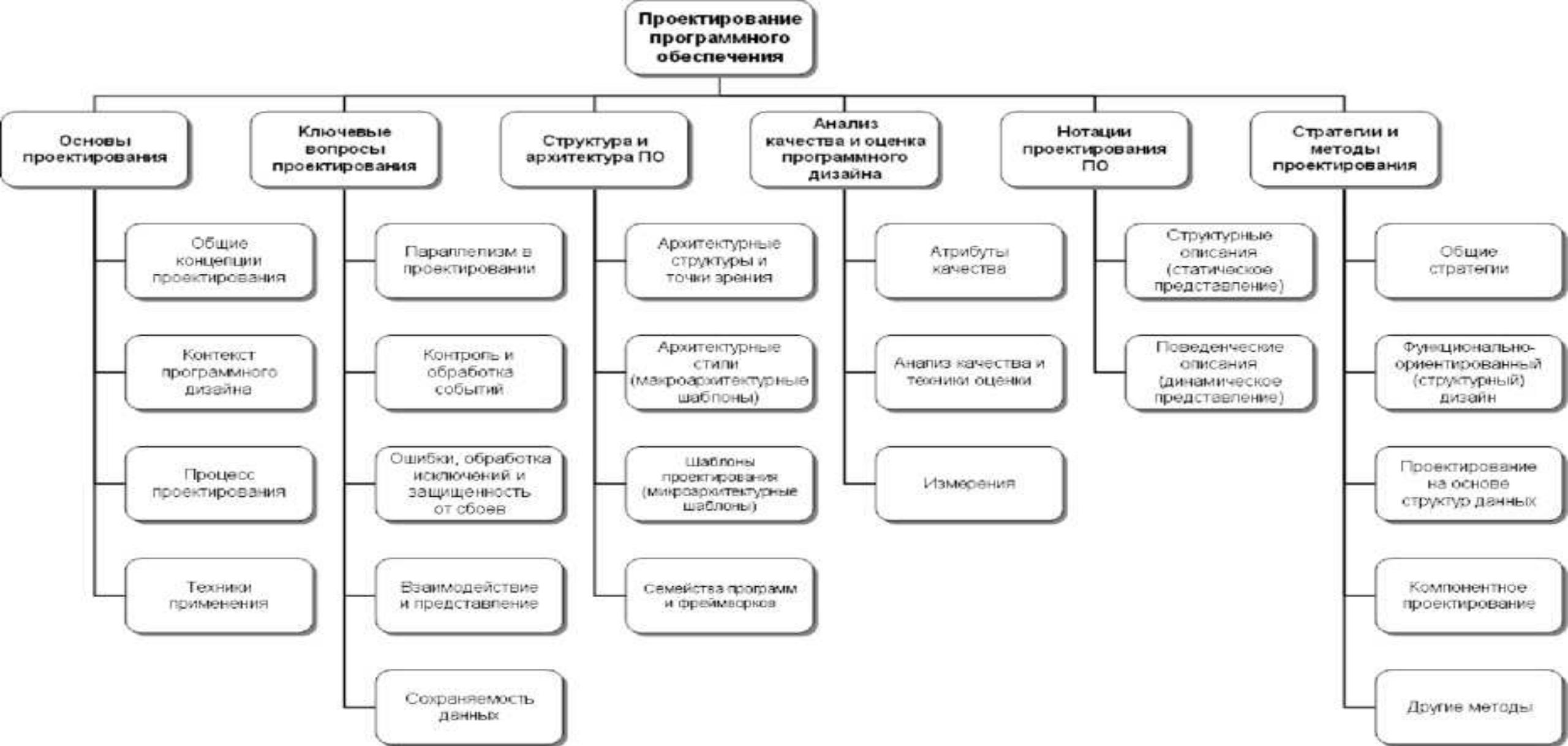
SWEBOK





Основы программных требований (Software Requirements Fundamentals)

Эта секция включает определение программных требований как таковых и описывает основные типы требований и их отличия: к продукту и процессу, функциональные и нефункциональные требования и т.п.



Основы проектирования (Software Design Fundamentals)

Эта секция вводит концепции, понятия и терминологию в качестве основы для понимания роли и содержания проектирования (как деятельности) и дизайна (архитектуры, как результата) программного обеспечения.



Основы конструирования (Software Construction Fundamentals)

Фундаментальные основы конструирования программного обеспечения включают:

- Минимизация сложности
- Ожидание изменений
- Конструирование с возможностью проверки
- Стандарты в конструировании

Первые три концепции применяются не только к конструированию, но и проектированию, и лежат в основе современных методологий управления жизненным циклом программных систем

Интеграция



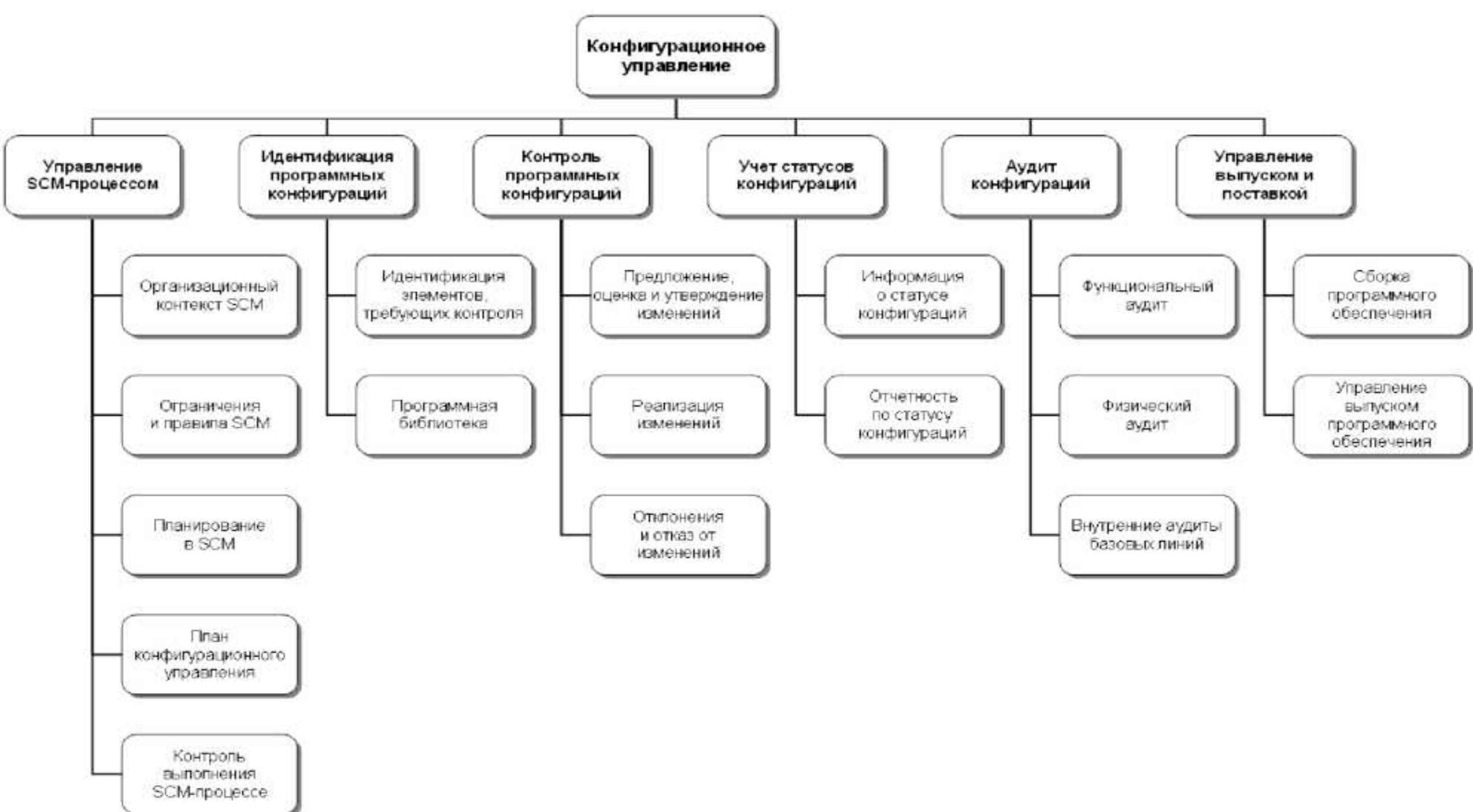
Основы тестирования (Software Testing Fundamentals)

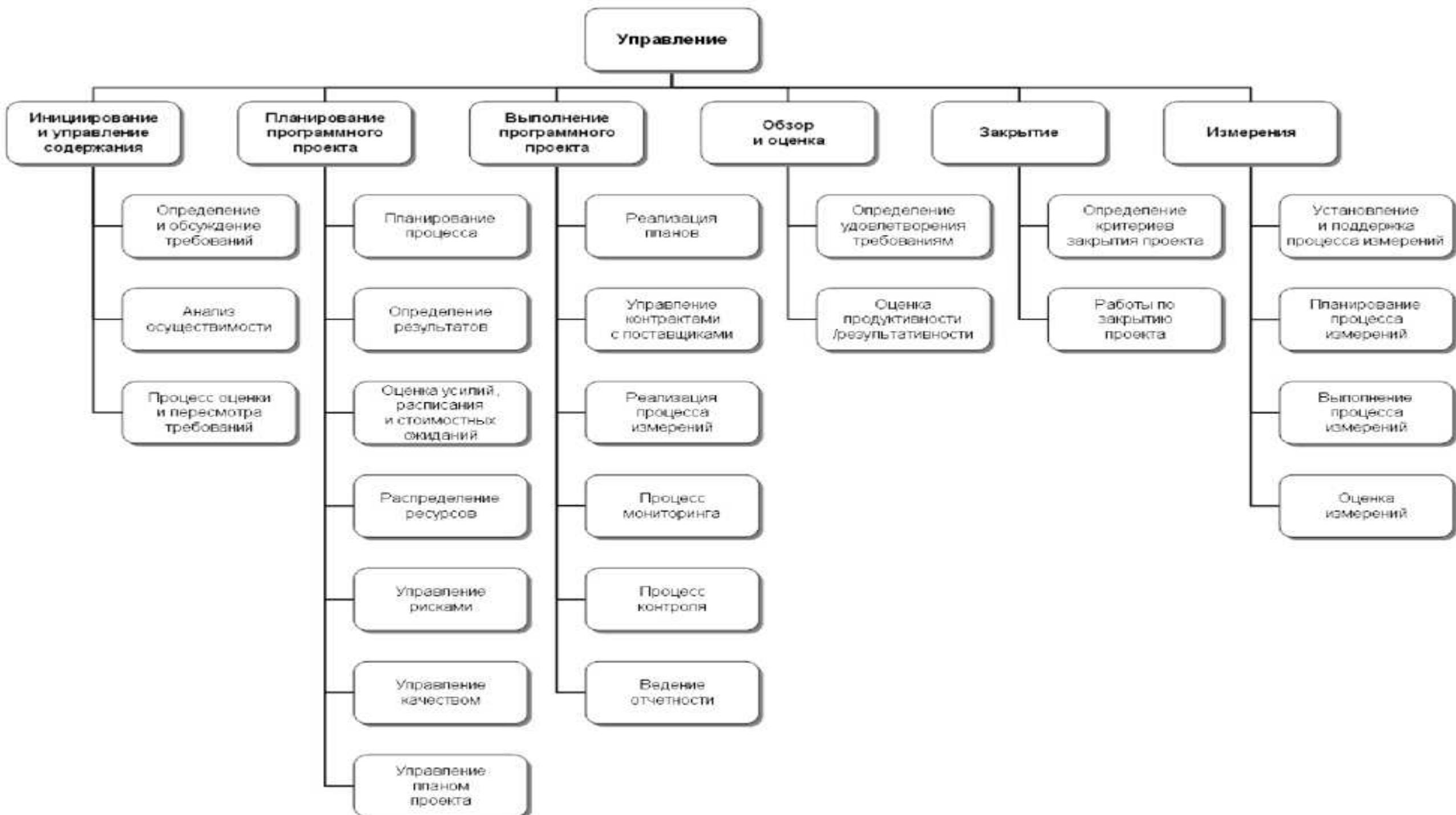
Охватывает основные понятия в области тестирования, базовые термины, ключевые проблемы и их связь другими областями деятельности и знаний.



Основы сопровождения программного обеспечения (Software Maintenance Fundamentals)

Эта секция включает концепции и терминологию, формирующие основы понимания роли и содержания работ по сопровождению программных систем. Темы данной секции предоставляют соответствующие определения и описывают, почему именно существует потребность в сопровождении. Категории сопровождения критически важны для понимания сути обсуждаемых вопросов.



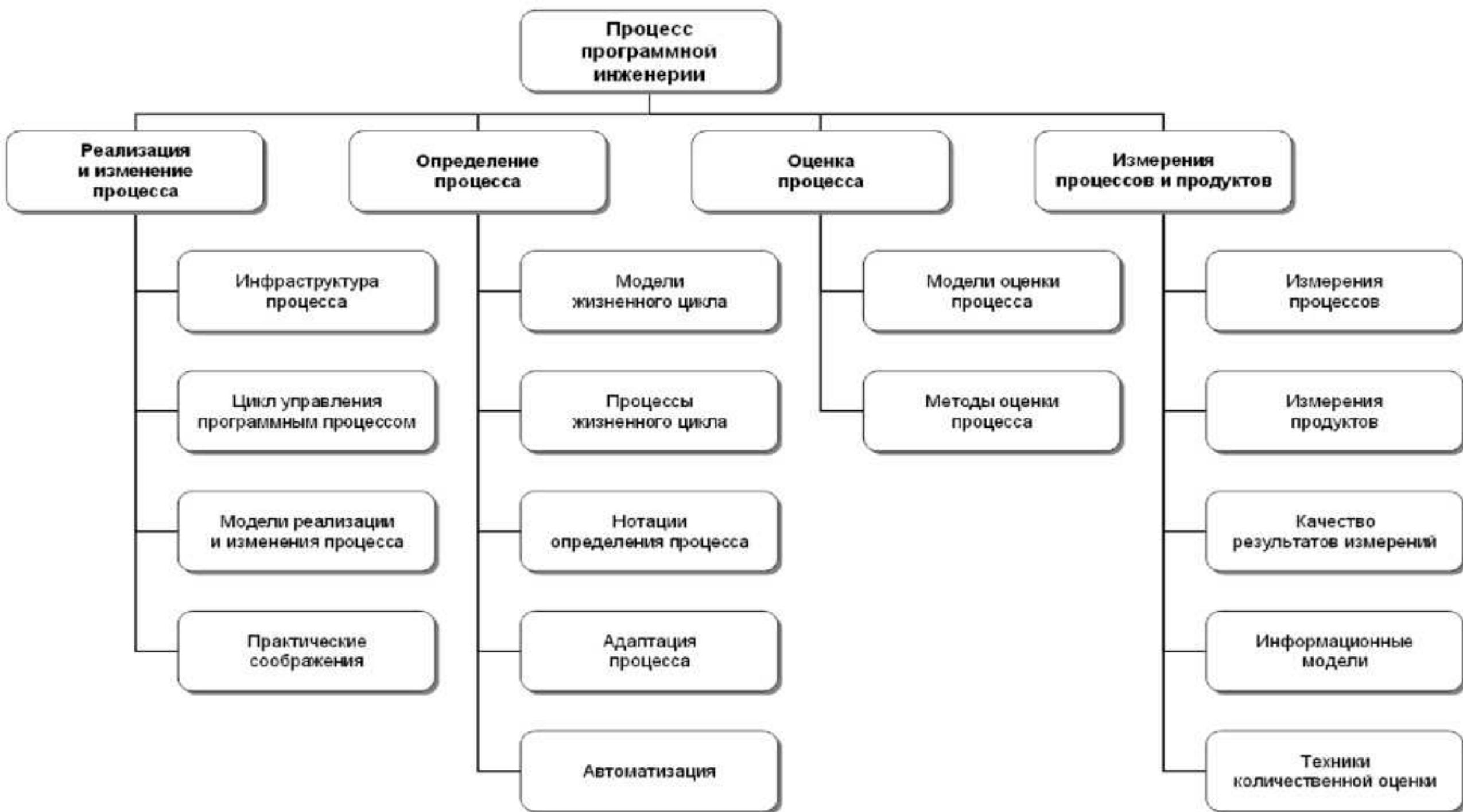


Управление программной инженерией может быть определено как приложение вопросов управления (management activities) - планирования, координации, количественной оценки, мониторинга, контроля и отчетности - к инженерной деятельности для систематического, упорядоченного и количественно измеряемого обеспечения разработки и сопровождения программных систем (IEEE 610.12-90, Standard Glossary for Software Engineering Terminology).

Таким образом, область знаний "Управление программной инженерией" определяет аспекты управления и количественной оценки в программной инженерии. Измерения являются важным аспектом для всех областей знаний SWEBOK и соответствующая тема также включена и в описании данной области знаний.

В принципе, корректно утверждать, что возможно управлять программной инженерией так же, как и любым другим комплексным процессом. В то же время, существуют аспекты, специфичные для программных продуктов, а процессы жизненного цикла программных систем, в какой -то мере, усложняют достижение необходимого уровня эффективности управления. Среди таких усложняющих факторов:

- Восприятие клиентов (потребителей, заказчиков) таково, что часто отсутствует с их стороны понимание сложности, порожденной самой природой программной инженерии, в частности связанной с влиянием изменяющихся требований.
- Практически неизбежно изменение или появление новых клиентских требований, как следствие функционирования процессов программной инженерии.
- В результате, программные системы часто строятся с применением итеративных процессов, вместо последовательного выполнения и закрытия работ (задач).
- Уровень новизны и сложности программных систем часто крайне высок (и не позволяет в полной мере применять уже существующие наработки и опыт).
- Применяемые технологии обладают высокой скоростью изменения, обновления и устаревания



Область знаний “Процесс программной инженерии” (Software Engineering Process) может быть рассмотрена на двух уровнях. Первый уровень содержит техническую и управленческую деятельность на протяжении процессов жизненного цикла программного обеспечения, включающих приобретение, разработку, сопровождение и вывод из эксплуатации программных систем. Второй уровень - “мета-уровень”, связанный с определением, реализацией, оценкой, измерением, управлением, изменением и совершенствованием самих процессов жизненного цикла программного обеспечения. Первый уровень освещен в других областях знаний SWEBOK. Второй уровень рассматривается в данной области знаний. Термин “процесс программной инженерии” (software engineering process) может интерпретироваться по-разному и это, соответственно, может приводить к определенной путанице.

- С одной стороны, учитывая специфику оригинального термина в английском языке, где (с точки зрения грамматики) может существовать термин the software engineering process, он будет подразумевать единственно правильный способ выполнения задач (performing tasks) программной инженерии. Такое предположение заведомо отбрасывается SWEBOK, так как “единственно правильного” процесса быть не может. Такие стандарты, как IEEE/ISO/ГОСТ 12207 говорят о процессах (во множественном числе - processes), подразумевая что программная инженерия содержит множество процессов, например, процесс разработки (Development Process) и процесс конфигурационного управления (Configuration Management Process).
- Вторая интерпретация связана с общим (general) обсуждением процессов, связанных с программной инженерией. Данная точка зрения отражена в названии этой области знаний и является одной из наиболее часто подразумеваемых при использовании термина “процесс программной инженерии”. уровне организации. Такой подход также рассматривается в данной области знаний (та или иная интерпретация обычно зависит от контекста обсуждения).

Инструменты и методы

Инструменты

Методы

Инструменты работы с требованиями

Инструменты проектирования

Инструменты конструирования

Инструменты тестирования

Инструменты сопровождения

Инструменты конфигурационного управления

Инструменты управления инженерной деятельностью

Инструменты поддержки процессов

Инструменты обеспечения качества

Дополнительные аспекты инструментального обеспечения

Эвристические методы

Формальные методы

Методы прототипирования

Программные инструменты предназначены для обеспечения поддержки процессов жизненного цикла программного обеспечения.

Инструменты позволяют автоматизировать определенные повторяющиеся действия, уменьшая загрузку инженеров рутинными операциями и помогая им сконцентрироваться на творческих, нестандартных аспектах реализации выполняемых процессов.

Инструменты часто проектируются с целью поддержки конкретных (частных) методов программной инженерии, сокращая административную нагрузку, ассоцииированную с “ручным” применением соответствующих методов. Так же, как и методы программной инженерии, инструменты призваны сделать программную инженерию более систематической деятельностью и по своему содержанию (предлагаемой функциональности) могут варьироваться от поддержки отдельных индивидуальных задач вплоть до охвата всего жизненного цикла (в этом случае часто говорят об инструментальной платформе или просто платформе разработки).

Методы программной инженерии накладывают определенные структурные ограничения на деятельность в рамках программной инженерии с целью приведения этой деятельности в соответствие с заданным систематическим подходом и более вероятным и скорым, с точки зрения соответствующего метода, достижением успеха.

Методы обычно предоставляют соответствующие соглашения (нотацию), словарь <терминов и понятий> и процедуры выполнения идентифицированных (и охватываемых методом) задач, а также рекомендации по оценке и проверке <выполняемого> процесса и <получаемого в его результате> продукта.

Методы, как и инструменты, варьируются по содержанию (охватываемой области применения) от отдельной фазы жизненного цикла (или даже процесса) до всего жизненного цикла.

Данная область знаний касается только методов, охватывающих множество фаз (этапов) жизненного цикла. Те методы, применение которых фокусируется на отдельных фазах жизненного цикла или частных процессах, описаны в соответствующих областях знаний.

Качество программного обеспечения

Основы качества

Процессы управления качеством

Практические соображения

Культура и этика
программной инженерии

Значение и стоимость
качества

Модели и характеристики
качества

Повышение качества

Подтверждение
качества

Проверка и аттестация
(V&V)

Оценка и аудит

Требования
к качеству

Характеристика
дефектов

Техника управления
качеством

Количественная
оценка качества

Что такое качество и почему оно должно быть столь глубоко представлено (в виде самостоятельной области знаний) в SWEBOK?

На протяжении многих лет отдельные авторы и целые организации определяли термин “качество” по-разному. Фил Кросби (Phil Crosby) в 1979 году дал определение качеству как “соответствие пользовательским требованиям”.

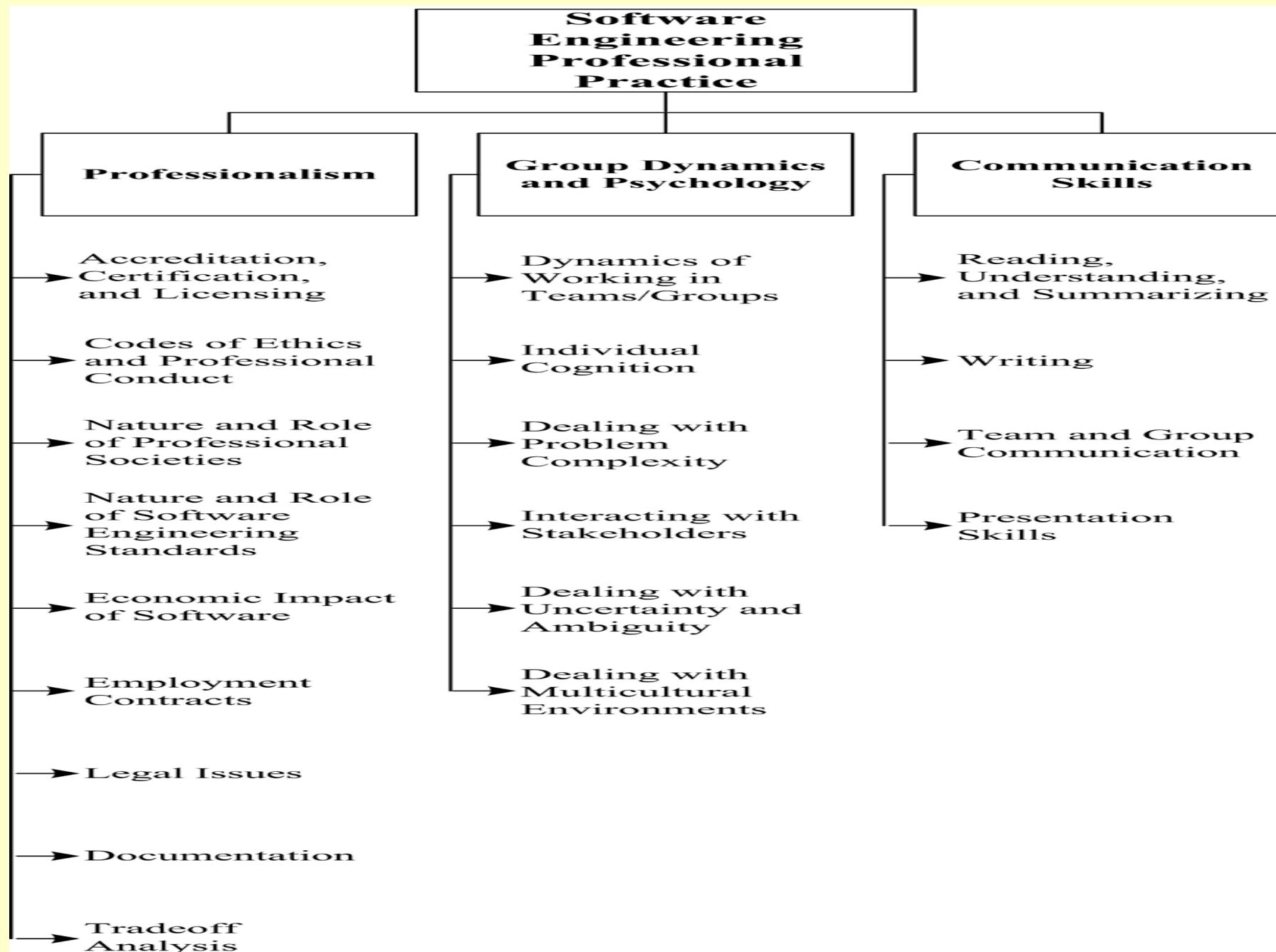
Уотс Хемпфри (Watts Humphrey, оригинальный автор концепции модели оценки зрелости СММ, а также PSP и TSP - People Software Process и Team Software Process) описывает качество как “достижение отличного уровня пригодности к использованию”.

Компания IBM, в свою очередь, ввела в оборот фразу “качество, управляемое рыночными потребностями” (“market-driven quality”).

Критерий Бэлдриджа (Baldrige) для организационного качества (см. NIST - National Institute of Standards and Technology, “Baldrige National Quality Program”, <http://www.quality.nist.gov>) использует похожую фразу - “качество, задаваемое потребителем” (“customer-driven quality”), рассматривая удовлетворение потребителя в качестве главного соображения в отношении качества.

Чаще, понятие качества используется в соответствии с определением системы менеджмента качества ISO 9001 как “степень соответствия присущих характеристик требованиям” (именно так это сформулировано в официальном переводе ИСО 9000-2000 “Системы менеджмента качества. Основные положения и словарь”).

Профессиональная практика разработки и сопровождения программного обеспечения



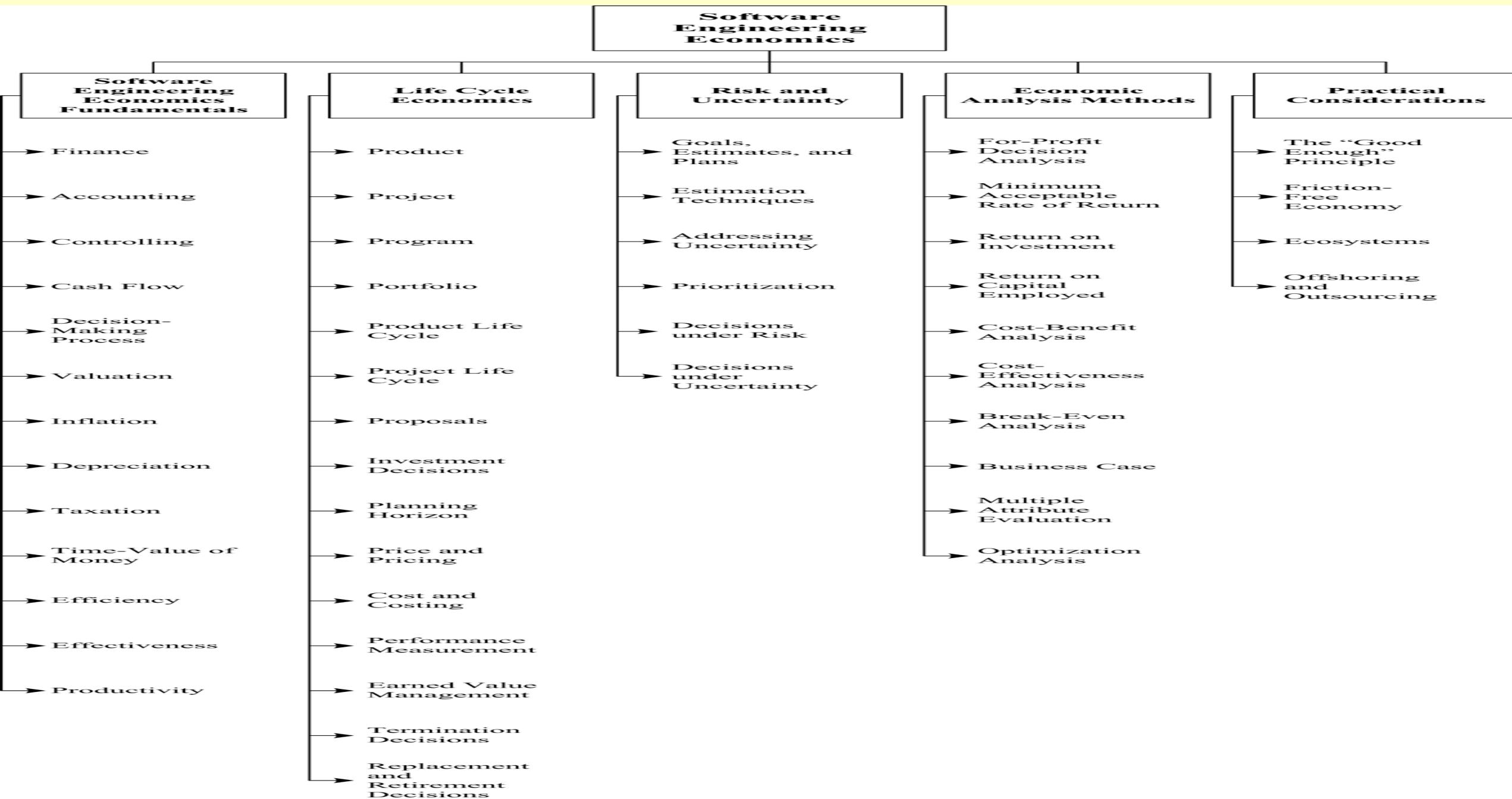
Профессиональная практика по Программной Инженерии область знаний связана с знаниями, навыками и отношением к программному обеспечению, которыми инженеры должны обладать при разработке программного обеспечения.

Из-за широко распространенных применений программных продуктов в социальной и личной жизни, качество программных продуктов может иметь глубокое влияние на наше личное благополучие и общественную гармонию. Разработчики программного обеспечения должны применять уникальный опыт инженерии, при разработке программного обеспечения с известными характеристиками и надежностью.

Это требование касается инженеров-программистов, которые обладают надлежащим набором знаний, умений и навыков обучения, опытом профессиональной практики.

Термин “профессиональная практика” относится к способу предоставления услуг таким образом, чтобы стандарты или критерии достигались при получении конечного продукта. Эти стандарты и критерии могут включать как технические, так и нетехнические аспекты.

Экономика разработки и сопровождения программного обеспечения



Экономика разработки и сопровождения программного обеспечения- это решения, связанные с разработкой программного обеспечения рассматриваемые в бизнес-контексте.

Успех программного продукта, обслуживание и решение зависят от хорошего управления.

Тем не менее, во многих компаниях и организациях, сопровождение и разработка программного обеспечения характеризуются не системным или слабо - системным подходом.

Эта область знаний обеспечивает обзор экономики программной инженерии.

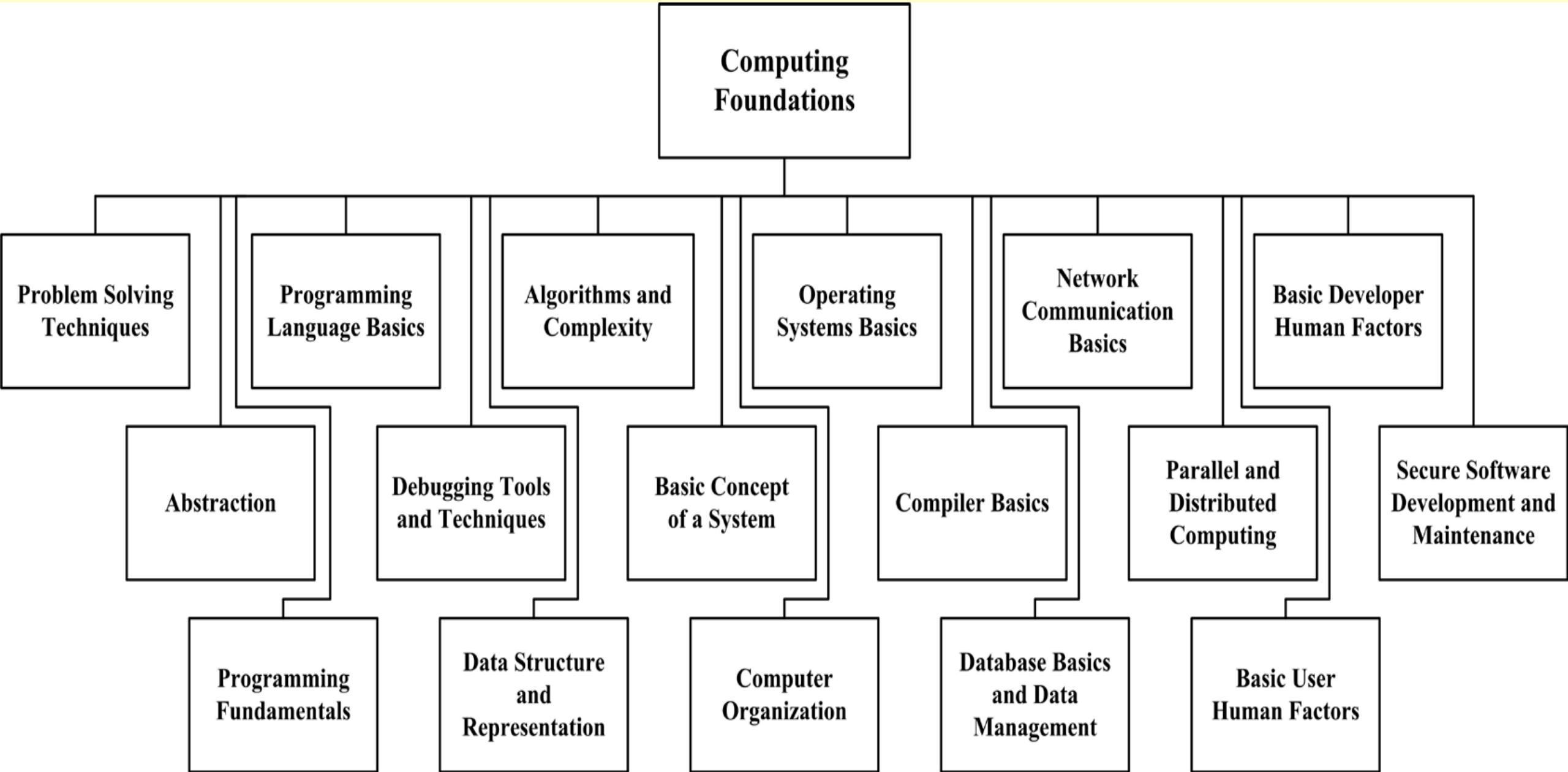
Экономика - это изучение стоимости, затрат, ресурсы и их взаимосвязь в данном контексте или ситуации.

По дисциплине программная инженерия, мероприятия имеют затраты, то есть программное обеспечение также имеет экономические атрибуты.

Экономика программной инженерии обеспечивает способ изучить атрибуты программного обеспечения и рассматривать разработку и сопровождение программного обеспечения, как процессы на систематической основе, которая их связывает с экономическими мерами.

Эти экономические меры могут быть взвешены и проанализированы при принятии решений в рамках соответствующей программы организации.

Вычислительные основы разработки и сопровождения программного обеспечения

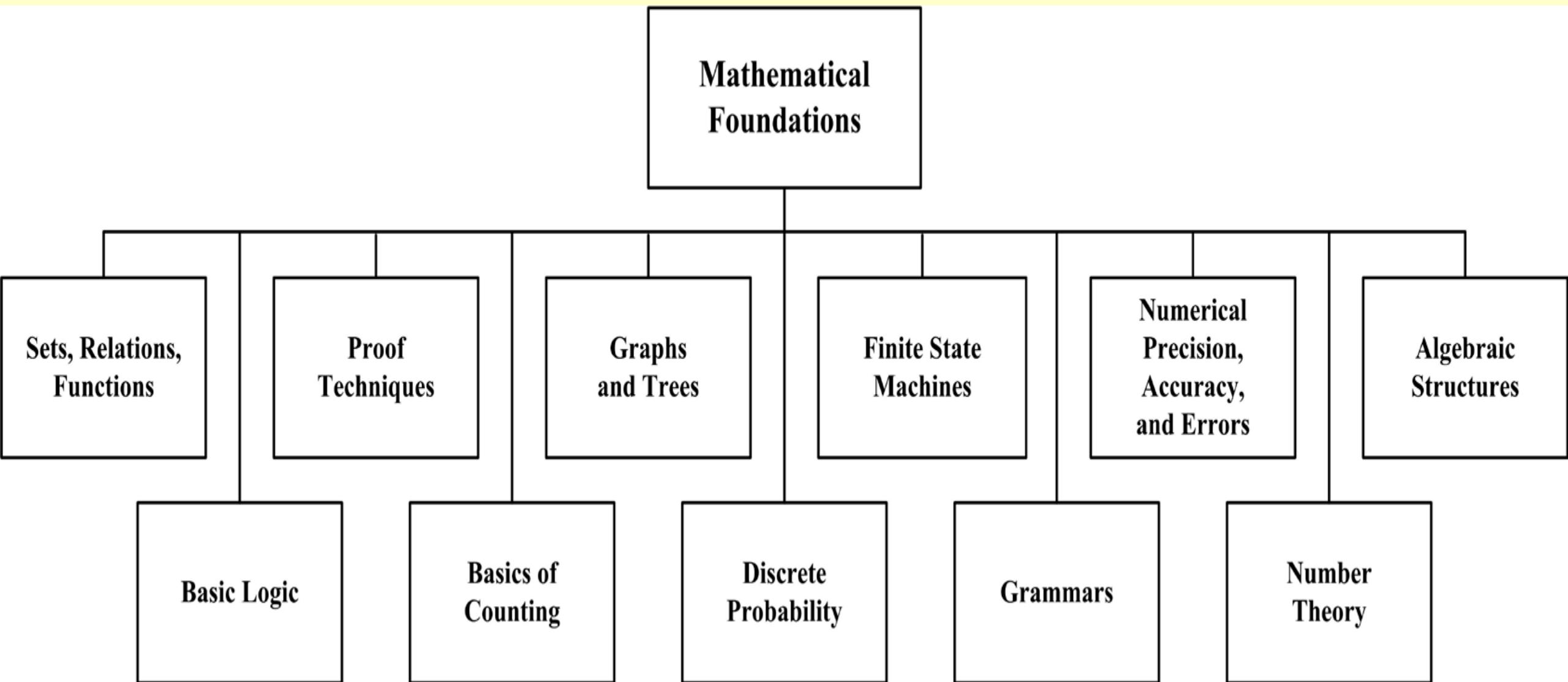


Вычислительные основы разработки и сопровождения программного обеспечения включает развитие рабочей среды, в которой программное обеспечение развивается и выполняется, потому что никакое программное обеспечение не может существовать в вакууме или работать без компьютера. Основой такой среды является компьютер и его различные компоненты. Знание о компьютере и о его основных принципах аппаратного и программного обеспечения, является основой на которой стоит программная инженерия.

Таким образом, все инженеры-программисты должны иметь хорошее понимание вычислительных основ современных вычислительных систем.

Общепризнано, что разработка программного обеспечения опирается на компьютерные науки. Один особенно важный аспект заключается в том, что разработка программного обеспечения основывается на информатике и математике.

Математические основы разработки и сопровождения программного обеспечения



Математические основы разработки и сопровождения программного обеспечения основывается на том , что профессионалы в области программного обеспечения “живут” с программами.

На языке программирования , можно запрограммировать только то, что хорошо понимаешь.

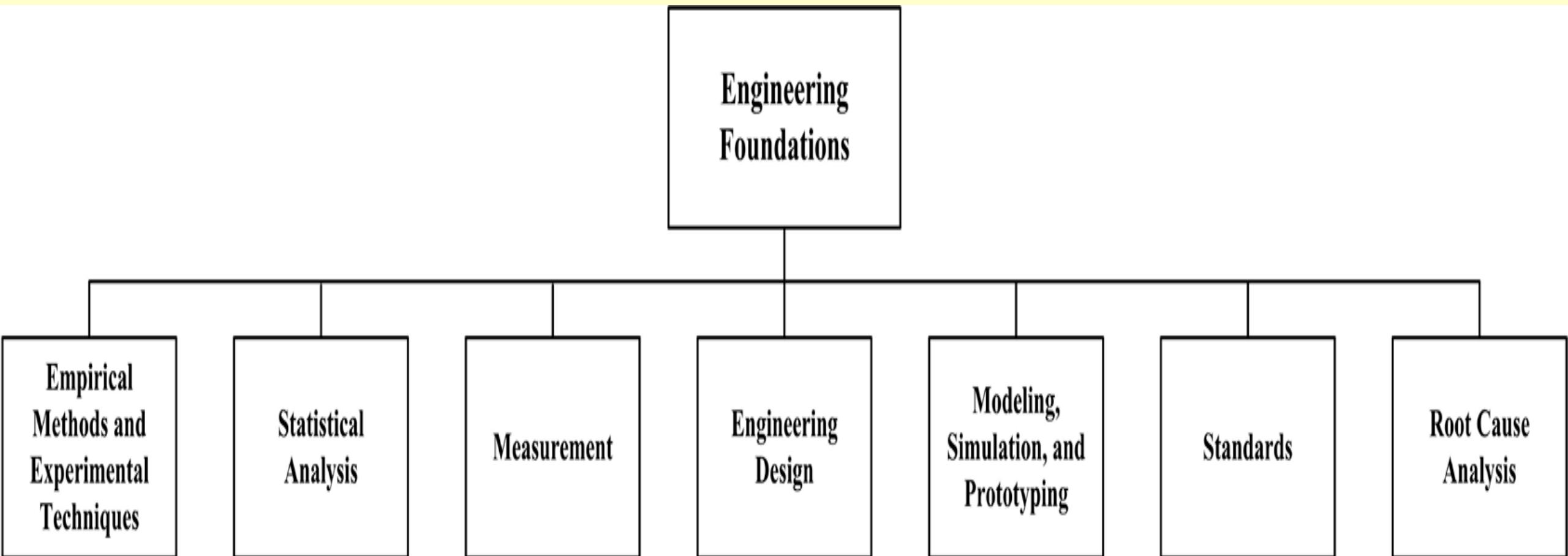
Логика - математическая основа области знаний, которая помогает разработчикам программного обеспечения понять эту логику, что в свою очередь переводится в код языка программирования.

Основной упор математики в программировании значительно отличается от обычной арифметики, где цифры рассматриваются и обсуждаются.

В программировании логика и рассуждения суть применения математики.

Программное обеспечение и инженер должны быть единым целым.

Инженерные основы, разработки и сопровождения программного обеспечения



IEEE определяет проектирование как “систематический, дисциплинированный и поддающийся количественной оценке подход к конструкциям, машинам, изделиям, системам или процессов”. В этой области знаний излагаются некоторые инженерные основополагающие навыки и методы ,что является полезным для инженера-программиста. Центр тяжести находится на темах, которые поддерживают другие области знаний при минимизации дублирования тем в другом месте этого документа.

Лекция

Семейства операционных

систем .

Понятие и назначение операционной системы.

Операционные системы – это программные комплексы предназначенные для решения трех задач:

- Автоматизация процесса распределения системных ресурсов.

Это обеспечивает эффективное использование аппаратного обеспечения. Поскольку на современных вычислительных машинах одновременно запускаются далеко не одна программа, то ОС отвечает за распределение памяти, регистров процессора и др. между запущенными программами в каждый момент времени. ОС определяет оптимальное распределение этих ресурсов во времени (использование процессора программами по очереди) и пространстве (загрузка в разные части оперативной памяти разных программ).

- Обеспечение интерфейса человек-вычислительная система и вычислительная система-вычислительная система.

Это облегчает (или даже предоставляет возможность) пользователям и программистам использование аппаратного обеспечения. Например, операционная система дает возможность абстрагироваться от того как на самом деле происходит обработка данных на жестком диске, а работать с понятием файла.

- Обеспечение безопасности информации

Системные ресурсы – время задействования CPU или APU, оперативная память, память на внешних устройствах HDD и SSD, клавиатура , монитор, USB контролеры, SATA – контролеры и т.д.

Важнейшим системным ресурсом являются порты-ввода вывода. Это обязательный ресурс для взаимодействия отдельных устройств с CPU или APU. Порты ввода-вывода обычно имеют адреса начиная с 0 до $2^{**}6$.

Второй по значимости ресурс – номер прерывания. Является не обязательным ресурсом, но позволяет значительно оптимизировать процессы в вычислительных системах

Так же операционные системы (ОС) можно отнести к системному ПО.

ОС обеспечивает реализацию алгоритмов работы с аппаратным обеспечением.

Может возникнуть вопрос: зачем это нужно?

Каждая прикладная программа может включать код, обеспечивающий обращение к «железу». Однако, это только бы усложнило жизнь программистам и раздудло бы ПО до больших размеров.

В прикладных программах было бы много одинакового кода, отвечающего за реализацию низкоуровневых команд (обращений к железу).

Кроме того, как решить проблему совместной работы разных программ на одном компьютере — еще один вопрос.

Поэтому операционные системы и другое системное ПО вполне обоснованно занимают отведенную им роль посредника между прикладным ПО и аппаратным обеспечением компьютера.

В своем историческом развитии операционные системы зародились именно как набор программ и библиотек для управления операциями ввода и вывода.

Этими достаточно универсальными программами далее пользовались остальные програмисты, которым уже не нужно было ломать голову как запрограммировать считывание данных с дискеты или вывод текста на принтер.

Они просто вызывали функцию из подключенной библиотеки, а она делала всю работу (в ней уже был заложен код работы с физическими устройствами).

С течением времени операционная система все более усложнялась, на нее возлагали новые **функции**.

Компьютеры становились мощнее, потребовалась одновременно запускать определенное множество программ на выполнение процессору. ОС стала решать задачи эффективного распределения ресурсов «железа» между работающими программами. С одной вычислительной машиной стали одновременно работать несколько пользователей. ОС стала следить за правами каждого и защищать данные. В результате современные ОС включают в себя множество различных функций.

По своему строению операционная система представляет комплекс программ и модулей.

Выделяют понятие ядра операционной системы.

Программное обеспечение ядра защищено от вмешательства пользователей и программистов.

К ядру прикладные программы обращаются с помощью запросов на выполнение того или иного действия с аппаратным обеспечением.

Эти запросы называются системными вызовами и представляют собой специальные команды.

Архитектура ОС.

Наиболее общим подходом к структуризации операционной системы является разделение всех ее модулей на две группы:

- 1. Ядро – модули, выполняющие основные функции ОС.
- 2. модули, выполняющие вспомогательные функции.

Модули ядра выполняют:

- · управление процессами
- · управление памятью
- · управление устройствами ввода-вывода

Ядро выполняет такие функции, как переключение контекстов, загрузка/выгрузка страниц памяти, обработку прерываний. Эти функции недоступны для приложений. Они создают для них так называемую **прикладную программную среду**. Приложения обращаются к ядру с запросами — системными вызовами — для выполнения тех или иных действий (например, открытия и чтения файла, вывода графики, получения системного времени и т. п.).

Функции ядра, которые могут вызываться приложениями, образуют **интерфейс прикладного программирования**.

Функции, выполняемые модулями ядра — наиболее часто используемые, поэтому скорость их выполнения определяет производительность всей системы в целом. Поэтому большая часть модулей ядра являются резидентными, т.е. постоянно находятся в оперативной памяти.

Вспомогательные модули ОС оформляются либо в виде приложений, либо в виде библиотек процедур.

Поскольку часть модулей ОС выполняются как обычные приложения (т.е. в стандартном для данной ОС формате), то часто бывает сложно провести четкую грань между ОС и приложениями.

Решение о том, является ли какая-либо программа частью ОС или нет, принимает производитель ОС.

Вспомогательные модули ОС подразделяются на следующие группы:

- утилиты — программы, решающие отдельные задачи управления и сопровождения компьютерной системы (например, программы сжатия дисков, архивирования данных);
- системные обрабатывающие программы — текстовые и графические редакторы, компиляторы, компоновщики, отладчики;
- программы предоставления пользователю дополнительных услуг — специальный пользовательский интерфейс, калькулятор и т. п.
- библиотеки процедур — упрощают разработку приложений (например, библиотека математических функций, функций ввода-вывода и т. п.).

Вспомогательные модули ОС обращаются к функциям ядра посредством системных вызовов.

Разделение операционной системы на ядро и модули-приложения обеспечивает легкую расширяемость ОС.

Вспомогательные модули ОС загружаются в оперативную память только на время выполнения своих функций, т.е. являются **транзитными**.

Важным свойством архитектуры, основанной на ядре, является возможность защиты кодов и данных операционной системы за счет выполнения функций ядра в привилегированном режиме.

Обеспечить привилегии ОС невозможно без специальных средств аппаратной поддержки. Аппаратура компьютера должна поддерживать как минимум два режима работы: пользовательский режим и привилегированный режим, его также называют режимом ядра или режимом супервизора.

Приложения ставятся в подчиненное положение за счет запрета выполнения в пользовательском режиме некоторых критичных команд, связанных с переключением процессора с задачи на задачу, управлением устройствами ввода-вывода, доступом к механизмам распределения и защиты памяти.

Условие разрешения выполнения критичных инструкций находятся под контролем ядра и обеспечивается за счет набора инструкций, безусловно, запрещенных для пользовательского режима.

Аналогично обеспечиваются привилегии ядра при доступе к памяти. Например, выполнение инструкции доступа к памяти для приложения разрешается, если инструкция обращается к области памяти, отведенной ОС данному приложению, и запрещается при обращении к другим областям. Между количеством уровней привилегий, реализуемых аппаратно, и количеством уровней привилегий, поддерживаемых ОС, нет прямого соответствия. Так, например, на базе четырех уровней, обеспечиваемых процессорами Intel, ОС OS/2 строит трехуровневую систему привилегий, а Windows NT – двухуровневую.

С другой стороны, если аппаратура поддерживает хотя бы два уровня, программным способом можно построить ОС со сколь угодно развитой системой защиты.

Рассмотренная архитектура ОС, основанная на привилегированном ядре и приложениях пользовательского режима, стала классической. Ее используют многие известные ОС: UNIX, VMS, OS/390, OS/2, Windows NT.

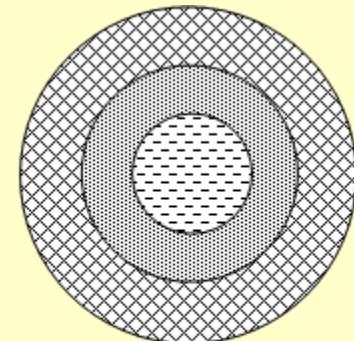
В некоторых случаях разработчики отступают от этой классики, и привилегированный режим используется и для приложений ОС.

В этом случае обращения приложений к ядру осуществляются быстрее, но при этом отсутствует надежная аппаратная защита памяти.

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.

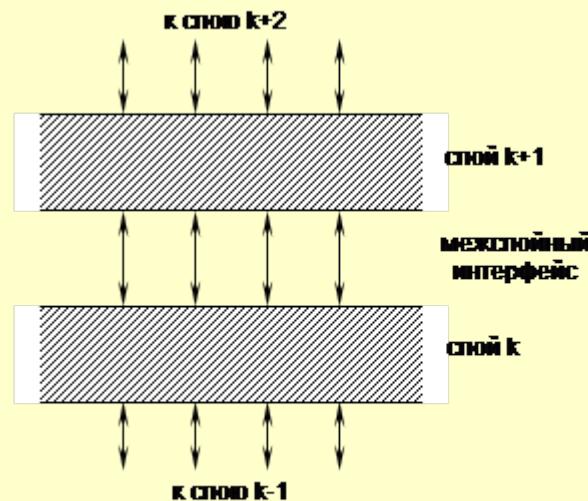


- вспомогательные функции
- ядро ОС
- аппаратура

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 вышеперечисленные слои ядра от особенностей аппаратуры. Это позволяет разрабатывать вышеперечисленные 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 — исполнительными механизмами 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 случаях 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.

Эксплуатационные требования, предъявляемые к ОС

Независимо от того, какие функции выполняет ОС, она должна удовлетворять эксплуатационным требованиям. Она, в частности, должна обладать следующими качествами:

- 1. **Надежностью.** В случае ошибки в программном или аппаратном оборудовании система должна обнаружить ошибку и либо попытаться исправить положения. Либо сообщить об этом пользователю и постараться свести к минимуму ущерб.
- 2. **Защитой.** Пользователь не хочет, чтобы другие пользователи (если он, например, работает в сети) ему мешали. Поэтому ОС должна защищать пользователя от воздействия чужих ошибок и от попыток злоумышленного вмешательства.
- 3. **Эффективностью.** ОС – довольно сложная программа, которая использует значительную часть ресурсов для своих собственных надобностей. Ресурсы, которые потребляют ОС, не поступают в распоряжение пользователя. Следовательно ОС должна быть, как можно более экономной. Кроме того, она должна управлять ресурсами пользователей так, что бы свести к минимуму время задержки и простоев.
- 4. **Предсказуемостью.** Требование, которое пользователь может предъявлять к системе, в большинстве случаев, непредсказуемы. Но пользователь предпочитает, что бы обслуживание не очень сильно менялось в течении продолжительного времени. В частности, ввода программы в машину пользователь должен иметь основанные на предыдущем опыте приблизительное представление о том, когда ему следует ожидать выдачи результатов.
- 5. **Удобством.** Все ясно, как и то, что универсальных удобств не существует. Здесь может идти речь об определенном классе задач.

ФУНКЦИИ СОВРЕМЕННЫХ ОС:

- **Распределение процессора.** В случае незамысловатой системы вся она распределяется как единый ресурс. Пользователь либо распоряжается машиной, либо ждет, когда она будет предоставлена в его распоряжение. Такую стратегию очень легко организовать, но она не будет эффективно использовать оборудование. Для того чтобы работать параллельно с процессором, можно сделать так, чтобы одна программа выполняла операции ввода-вывода, пока другая занимает главный процессор. Хотя реализация такого подхода – более сложна, налицо преимущество – каждое устройство используется более интенсивно.
- **Управление памятью.** Управление памятью тесно связано с распределением процессора. Программы могут работать только тогда, когда они находятся в оперативной памяти, но не обязательно их держать там, если надежда получить процессор – невелика. В этом случае окажется, что память, которую они занимают – пропадает зря.

Оперативная память – это тоже распределяемый ресурс. Поэтому система расходует время для того чтобы расположить информацию рационально, стараясь держать полезные программы в оперативной памяти и уничтожать “свободные промежутки” между программами. Для этого система может использовать перемещение программ. Это делается для того, чтобы уменьшить объем бесполезно используемой памяти. Перемещение легче осуществить, если использовать специальные стратегии организации памяти. Эти стратегии позволяют ОС весьма гибко регулировать обмен информацией между оперативной и вспомогательной памятью.

- **Управление контролерами и внешними устройствами.** Методы распределения устройств ввода-вывода и каналов связи различны. Задача пользуется периферийным устройством столько времени, сколько ей нужно, а затем отказывается от него. Устройство с быстрым произвольным доступом, такие как, накопители на дисках, можно совместно использовать в нескольких задачах по принципу “операция за операцией”, т.е. некоторым задачам разрешается использовать устройство попаременно. Если, скажем, две задачи попытаются выполнить операцию ввода-вывода на одном устройстве одновременно, то возникают очередь и задержка.

Подход к распределению устройств оказывает существенное влияние как на правильность, так и на эффективность работы.

Стратегия распределения влияет и на эффективность использования устройств, например, того же диска.

Эффективное распределение периферийных устройств трудно реализовать по двум причинам:

- *Во-первых*. с помощью существующих математических методов нельзя провести необходимые исследования и отыскать оптимальные способы распределения нескольких различных типов устройств для общего случая решения задач.
- *Во-вторых*, эффективность стратегии распределения очень трудно измерить.

Можно оценить только общее внешнее проявление неправильного распределения. Если же при оценке учитывать и влияние взаимодействия с оперативной памятью и центральным процессором, то аналитические и эмпирические методы оказываются еще менее перспективными.

Методы распределения устройств ввода-вывода, контроллеров и каналов сильно зависит от устройств.

- **Управление программными ресурсами.** Часто в операционных системах имеются системы прикладных программ и библиотеки программ пользователей. Будучи ресурсами, подлежащими распределению, эти библиотеки имеют много общего с аппаратными ресурсами.

Совместное использование интерпретаторов, редакторов текстов и т.п. можно организовать, если эти программы допускают параллельное использование.

Если у них рабочая часть полностью отделена от данных и операций записи в память применяются только к разделу данных, то такие программы допускают параллельное использование. Если же программа не допускает параллельного использования, то каждый ее экземпляр может быть в данный момент только одному пользователю, так же, как и любое аппаратное устройство.

При параллельном использовании каждому пользователю выделяется личный экземпляр раздела данных, а с единственным экземпляром рабочей части все пользователи могут работать в режиме разделения.

- **Обеспечение интерфейса** человек-вычислительная система и вычислительная система-вычислительная система

Супервизор.

Традиционный подход при проектировании ОС состоит в том, что множество процессов, выполняющих основные функции системы, подчиняются главной программе, которая называется супервизором. Осуществляя централизованное управление, супервизор связывает воедино остальные части системы. Он организует совместную работу программ, устанавливая привилегии или назначая наказание. Он обеспечивает средства связи и синхронизации между процессами и физическими устройствами. Обычно и сообщения, передаваемые от процесса к процессу, и запуск и окончание работы устройств и сигнала от оборудования обрабатываются супервизором.

Супервизор, обычно, управляет разделением всех ресурсов у услуг системы между пользователями.

Функции супервизора. Функции супервизора можно грубо разделить на четыре области:

- контроль и управление;
- организация связей;
- защита и ограничения;
- обслуживающие программы;

Контроль и управление. В обязанности супервизора входит установление последовательности и контроль управления заданий в системе. Для обработки заданий служат планирование порядка выполнения заданий, учет потребления или ресурсов и интерпретация языка управления заданиями. Эти функции можно реализовать независимо от супервизора или подчинить ему.

Пользователь описывает свои требования на некотором языке управления заданиями. Супервизор интерпретирует эти требования и сообщает различным программам, распределяющим ресурсы, какие ресурсы и задачи запрошены заданиями. Распределители ресурсов обслуживаются запросы, возможно, в порядке их приоритетов. Программа учета запоминает количество потребленных ресурсов.

Организация связи. В супервизоре предусмотрены средства связи между различными программами, когда две независимые программы такие, как файловая система и программы управления памятью хотят связаться друг с другом, они должны просить супервизор установить контакт. После установления первоначального контакта программам разрешается передавать друг другу сообщения, принятым в данной системе способом, скажем, с помощью "писем" и общего "почтового ящика". Первоначальная связь устанавливается через супервизор. Доступная только супервизору информация позволяет ему также устанавливать контакты между системными программами и программами пользователей.

Защита и ограничения. Для обеспечения гарантий выполнения работы супервизор должен наложить некоторые ограничения, как на систему, так и на пользователей. Имеется ряд программ, обрабатывающих сигналы аппаратуры, особенно отклонения от нормальных условий. Например, во время работы программы копирования может возникнуть несколько десятков различных особых ситуаций, например, сбой при чтении или записи, неготовность дисководов к чтению или записи, отсутствие места на диске для копированного файла и т.д.

Когда поступает сигнал об ошибке (обычно это прерывание), супервизор должен принять решение задержать, повторить или даже выбросить задание. Для всех этих ситуаций необходимо предусмотреть соответствующие сообщения и корректирующие действия.

Ограничения должны налагаться на время работы программы и количество выдаваемых ею результатов.

Супервизор может также организовать систему защиты с помощью специальных “паролей”.

Доступ к защищенным ресурсам разрешается только по соответствующему паролю.

С целью защиты супервизора в некоторых системах существуют два типа режима работы:

- режим супервизора;
- рабочий режим.

Благодаря такой предосторожности супервизор обладает особой властью над привилегированными частями системы и программами пользователей. Супервизор может защищать себя, храня жизненно важную информацию в защищенной области памяти, доступной только в режиме супервизора.

Обслуживание программы. Помимо распределения ресурсов супервизор выполняет некоторые из функций системы.

В нем имеется набор обслуживающих программ для аварийных сборов, для доступа к библиотекам программ, для обработки сообщений от программ, работающих в оперативном режиме. Внутри супервизора часто находятся область памяти, предназначенная для организации связей и таблицы системы защиты. В ряде случаев, когда требуется выполнить сложные действия, супервизор вызывает специальные обрабатывающие программы.

Использование супервизора – это пример построения ОС на принципе централизованного управления.

Идеология централизации ключевых функций системы под контролем супервизора имеет свои преимущества и недостатки.

Одно из преимуществ – простая реализация защиты.

Второе преимущество – простота реализации.

С точки зрения разработки легче сосредоточить важнейшие системные функции в подчинении супервизора, чем распределять их по всей системе.

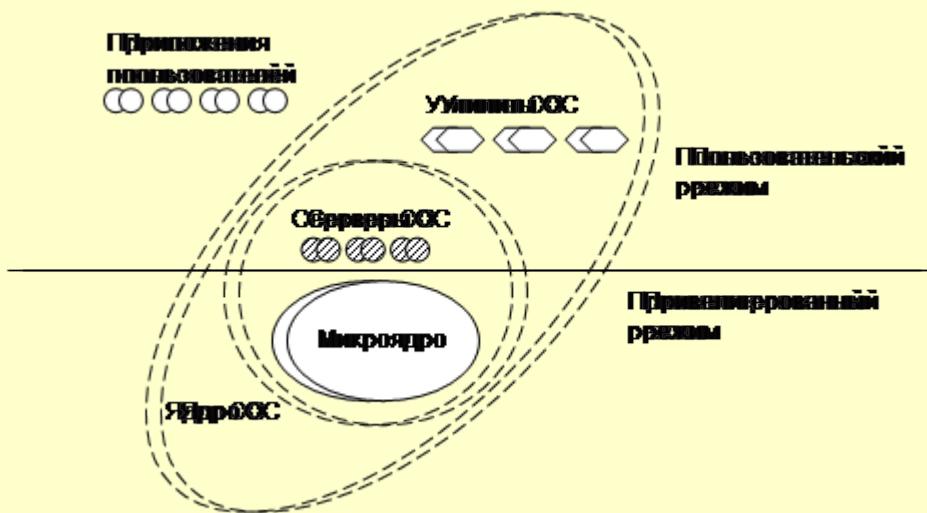
Лучше всего такой подход оправдывает себя, когда супервизор делают сравнительно небольшим. К сожалению, велико искушение передать супервизору очень много функций, превращая его тем самым собственно в ОС.

У централизованного супервизора есть и существенные недостатки. Так программам не разрешается устанавливать связи друг с другом самостоятельно.

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. Машино-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

Остальные функции ядра оформляются в виде приложений, работающих в пользовательском режиме. В общем случае многие менеджеры ресурсов, выполняются в виде модулей, работающих в пользовательском режиме.

Эти менеджеры, однако, имеют принципиальное отличие от традиционных утилит и обрабатывающих программ ОС, хотя при микроядерной архитектуре все эти программные компоненты также оформлены в виде приложений. Утилиты и обрабатывающие программы вызываются в основном пользователями. По определению, основным назначением такого приложения является обслуживание запросов других приложений, поэтому менеджеры ресурсов, вынесенные в пользовательский режим, называются **серверами ОС**.

Для реализации микроядерной архитектуры необходимым условием является наличие в ОС удобного и эффективного способа вызова одного процесса из другого. Поддержка такого механизма – одна из главных задач микроядра.

Преимущества микроядерной архитектуры

- Высокая переносимость;
- Расширяемость;
- Высокая надежность;
- Хорошие предпосылки для поддержки распределенных приложений, так как используются механизмы, аналогичные сетевым-взаимодействие клиентов и серверов путем обмена сообщениями. Серверы микроядерной ОС могут работать как на одном, так и на нескольких процессорах.

Недостатки микроядерной архитектуры

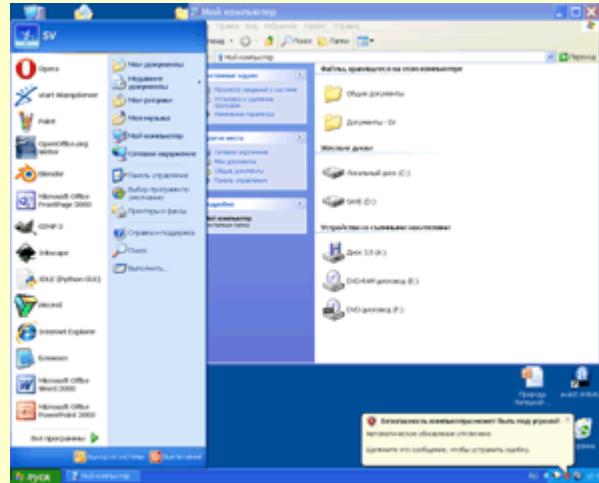
Более низкая производительность. Это сказывается на скорости работы прикладных сред, а значит и на скорости выполнения приложений.

Вывод:

- Микроядерная аппаратура является альтернативой классическому способу построения ОС, который предусматривает выполнение своих основных функций ОС в привилегированном режиме.
- В микроядерных ОС в привилегированном режиме остается работать очень небольшая часть ядра, которая названа микроядром. Все остальные функции ядра оформляются в виде приложений, работающих в пользовательском режиме.
- Микроядерные ОС удовлетворяют большинству требований, предъявляемых к совершенным ОС, обладая переносимостью, расширяемостью, надежностью и создают хорошие возможности для поддержки распределенных приложений.
- За эти достоинства приходится расплачиваться снижением производительности и это основной недостаток микропроцессорной архитектуры.
- Микроядерную концепцию используют такие ОС, как Windows NT и некоторые версии ОС UNIX.

Семейства операционных систем

ОС семейства Windows



На сегодняшний день наиболее популярными являются операционные системы семейства Windows, которые являются проприетарным (коммерческим) продуктом корпорации Microsoft.

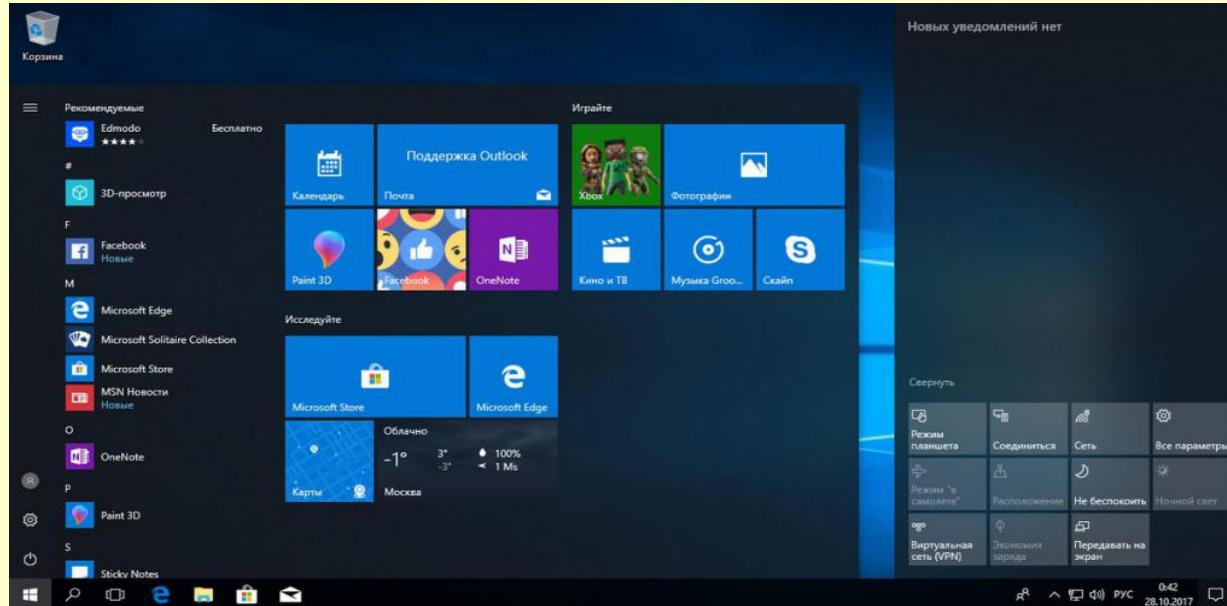
Свою «родословную» Windows начинают от операционной системы DOS и первоначально представляли собой надстраиваемые над ней оболочки (Windows запускался из под DOS), увеличивающие возможности DOS и облегчающие неподготовленному пользователю работу с компьютером. Уже более поздние версии (начиная с Windows NT) представляли собой полноценные операционные системы.

Преимуществом Windows считается дружественный для пользователя интерфейс. Из недостатков отмечают ненадежность системы.

Windows 10

Новейшая на сегодня версия операционной системы вышла в свет в июле 2015 года. Вот ее ключевые отличия от предыдущих:

- Модификация меню "Пуск": представлено в виде настраиваемых пользователем плиток.
- Изменение размера "Пуска".
- Новые возможности использования магазина приложений. Появление "Центра уведомлений".
- Обновленный календарь, часы, батарейный индикатор (для ноутбуков).
- Современные окна с новой анимацией.
- Обновленные интерфейсы приветствия и блокировки.



Семейство Unix-подобных операционных систем

Операционная система UNIX оказала большое влияние на развитие мира операционных систем, заложив основы работы современных ОС.

Изначально UNIX был системой для разработки ПО.

В основном в UNIX работали программисты (да и вообще в 70-е годы мало кто другой работал с вычислительными машинами).

UNIX развивался на нескольких фундаментальных идеях.

Например, одна небольшая задача должна решаться одной небольшой программой, а сложные задачи должны быть решаемы комбинацией простых программ.

В UNIX большое вниманиеделено распределению ресурсов компьютера между пользователями.

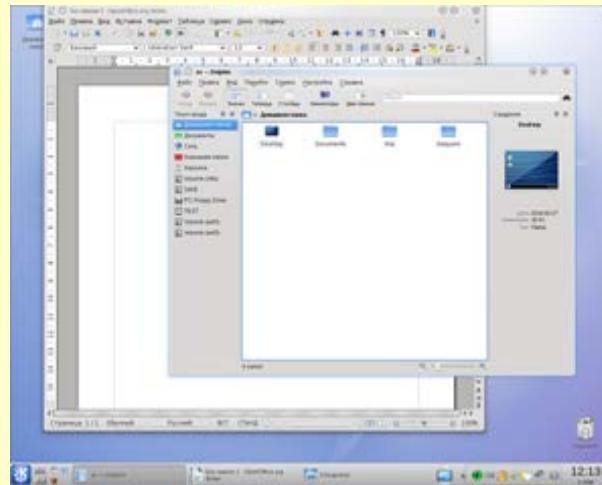
Эта система является мультитерминальной (каждый пользователь работает с компьютером с помощью своего терминала).

Не смотря на то, что Unix-подобные системы уступают по популярности Windows, они работают на больших типах компьютеров.

Семейство операционных систем UNIX уникально по нескольким причинам:

- оно является долгожителем и, претерпев многочисленные изменения, "завоевало" разнообразную аппаратуру;
- при переходе UNIX на другие аппаратные платформы возникали интересные задачи, решение которых принесло много нового в компьютерные технологии;
- на одной из версий UNIX были реализованы протоколы обмена данными в компьютерных сетях с разной аппаратной платформой, что позволяет считать UNIX предвестницей сегодняшнего Интернета, а также основой для широкого развития локальных сетей;
- авторы ее первых версий создали язык программирования высокого уровня C, который можно назвать (с учетом его последующего совершенствования) самым распространенным среди разработчиков;
- использование этого языка дало возможность принять участие в разработке операционной системы тысячам специалистов;
- появившиеся в семействе UNIX свободно распространяемые операционные системы внесли много нового в представление о том, как разрабатывать и распространять программы для компьютеров

Linux



Linux представляет собой множество Unix-подобных операционных систем (дистрибутивов), которые чаще всего являются свободно распространяемыми.

Одной из уникальных особенностей систем GNU/Linux является отсутствие единого географического центра разработки. Linux и программы для нее пишутся миллионами программистов, рассредоточенных по всему миру.

MAC OS



Это операционная система также создавалась на основе ядра UNIX. Является продукт компании Apple для ее же компьютеров Macintosh. Считается надежной и удобной. Но в отличие от Windows не так популярна.

Семейство операционных систем IBM

Операционная система OS/2

Сегодняшняя OS/2 - это мощная многозадачная операционная система с оконным графическим интерфейсом и набором созданных специально для нее прикладных программ, ориентированная на рынок персональных компьютеров и рабочих станций.

Интерфейс OS/2 включает все необходимые элементы современных OS - рабочий стол и корзину, иконки и панель задач, программу просмотра содержимого дисков, часы и драйвера множества периферийных устройств, таких как, например, порты USB или инфракрасный порт.

Инсталляция платформы производится автоматически, причем OS/2 самостоятельно определяет оптимальную конфигурацию системы исходя из быстродействия процессора и объема оперативной памяти (однако пользователь может и самостоятельно указать комплект необходимых программ, исключив ненужные), тестирует оборудование и настраивает все необходимые драйвера без участия оператора.

В комплект поставки входит пакет IBM Works, аналогичный MS Office и содержащий текстовый и табличный редактор, имеется удобный web-браузер WebExplorer и почтовый клиент NotesMail, система для создания анимации NeonGraphics, широчайший выбор всевозможных бизнес- приложений и множество игр от Civilisation и Quake III до Master of Orion.

Имеются и глобальные отличия OS/2 от привычной пользователям IBM PC Microsoft Windows - например, специальный самообучающийся программный пакет позволяет управлять системой с использованием голосовых команд, для чего в коробку с компакт-дисками разработчики вкладывают микрофон и наушники.

Однако, несмотря на поистине широчайшие возможности, высокую производительность и потрясающую надежность данной платформы, она не пользуется сейчас высоким спросом в силу доминирования на рынке более распространенной и дешевой MS Windows.

Основная проблема, препятствующая развитию OS/2, заключается в ее несовместимости с программами производства Microsoft, посредством которых создается практически вся деловая документация и с которыми работает подавляющее большинство частных пользователей.

Основные функции современных операционных систем

По современным представлениям ОС должна уметь делать следующее.

- Обеспечивать загрузку пользовательских программ в оперативную память и их исполнение (этот пункт не относится к ОС, предназначенным для прошивки в ПЗУ).
- Обеспечивать управление памятью. В простейшем случае это указание единственной загруженной программе адреса, на котором кончается память, доступная для использования, и начинается память, занятая системой. В многопроцессных системах это сложная задача управления системными ресурсами.
- Обеспечивать работу с устройствами долговременной памяти, такими как магнитные диски, ленты, оптические диски, флэш-память и т. д. Как правило, ОС управляет свободным пространством на этих носителях и структурирует пользовательские данные в виде файловых систем.
- Предоставлять более или менее стандартизованный доступ к различным периферийным устройствам, таким как терминалы, модемы, печатающие устройства или двигатели, поворачивающие рулевые плоскости истребителя.
- Предоставлять некоторый пользовательский интерфейс. Слово некоторый здесь сказано не случайно — часть систем ограничивается командной строкой, в то время как другие на 90% состоят из интерфейсной подсистемы. Встраиваемые системы часто не имеют никакого пользовательского интерфейса.

Существуют ОС, функции которых этим и исчерпываются. Одна из хорошо известных систем такого типа — **дисковая операционная система MS DOS**.

Более развитые ОС предоставляют также следующие возможности:

- параллельное (или псевдопараллельное, если машина имеет только один процессор) исполнение нескольких задач;
- организацию взаимодействия задач друг с другом;
- организацию межмашинного взаимодействия и разделения ресурсов;
- защиту системных ресурсов, данных и программ пользователя, исполняющихся процессов и самой себя от ошибочных и зловредных действий пользователей и их программ;
- аутентификацию (проверку того, что пользователь является тем, за кого он себя выдает), авторизацию (проверка, что тот, за кого себя выдает пользователь, имеет право выполнять ту или иную операцию) и другие средства обеспечения безопасности.

ОС общего назначения

Здесь под ОС мы будем подразумевать системы "общего назначения", т. е. рассчитанные на интерактивную работу одного или нескольких пользователей в режиме разделения времени, при не очень жестких требованиях ко времени реакции системы на внешние события.

Как правило, в таких системах уделяется большое внимание защите самой системы, программного обеспечения и пользовательских данных от ошибочных и злонамеренных программ и пользователей.

Обычно подобные системы используют встроенные в архитектуру процессора средства защиты и виртуализации памяти. К этому классу относятся такие широко распространенные системы, как ОС MS Windows, системы семейства Unix.

Системы реального времени

Это системы, предназначенные для облегчения разработки так называемых приложений *реального времени* — программ, управляющих некомпьютерным оборудованием, часто с очень жесткими ограничениями по времени.

Примером такого приложения может быть программа бортового компьютера fly-by-wire (дословно - "летящий по проволоке", т. е. использующий систему управления, в которой органы управления не имеют механической и гидравлической связи с рулевыми плоскостями) самолета, системы управления ускорителем элементарных частиц или промышленным оборудованием.

Подобные системы обязаны поддерживать многопоточность, гарантированное время реакции на внешнее событие, простой доступ к таймеру и внешним устройствам.

Способность гарантировать время реакции является отличительным признаком систем РВ.

Важно учитывать различие между гарантированностью и просто высокой производительностью и низкими накладными расходами.

Далеко не все алгоритмы и технические решения, даже и обеспечивающие отличное среднее время реакции, годятся для приложений и операционных систем РВ.

Эти системы могут относиться как к классу ДОС (RT-11), так и к ОС (OS-9, QNX).

Любопытно, что новомодное течение в компьютерной технике — multimedia — при качественной реализации предъявляет к системе те же требования, что и промышленные задачи реального времени.

В multimedia основной проблемой является синхронизация изображения на экране со звуком. Именно в таком порядке. Звук обычно генерируется внешним аппаратным устройством с собственным таймером, и изображение синхронизируется с ним. Человек способен заметить довольно малые временные неоднородности в звуковом потоке, а пропуск кадров в визуальном потоке не так заметен. Расхождение же звука и изображения фиксируется человеком уже при задержках около 30 мс.

Поэтому системы высококачественного multimedia должны обеспечивать синхронизацию с такой же или более высокой точностью, что мало отличается от реального времени.

Так называемое "*мягкое реальное время*" (*soft real time*), предоставляемое современными Win32 платформами, не является реальным временем вообще, это что-то вроде "осетрины второй свежести". Система "мягкого РВ" обеспечивает не гарантированное, а всего лишь среднее время реакции. Для мультимедийных приложений и игр различие между "средним" и "гарантированным" не очень критично — ну дернется картинка, или поплынет звук. Но для промышленных приложений, где необходимо настоящее реальное время, это обычно неприемлемо.

Средства кроссразработки

Это системы, предназначенные для разработки программ в двухмашинной конфигурации, когда редактирование, компиляция, а зачастую и отладка кода производятся на инструментальной машине (в англоязычной литературе ее часто называют *host* — дословно, "хозяин"), а потом скомпилированный код загружается в целевую систему.

Чаще всего они используются для написания и отладки программ, позднее прошиваемых в ПЗУ. Примерами таких ОС являются системы программирования микроконтроллеров Intel, Atmel, PIC и др., системы Windows CE, Palm OS и т. д.

Такие системы, как правило, включают в себя:

- набор компиляторов и ассемблеров, работающих на инструментальной машине с "нормальной" ОС;
- библиотеки, выполняющие большую часть функций ОС при работе программы (но не загрузку этой программы!);
- средства отладки.

Иногда встречаются кросс-системы, в которых компилятор работает не на инструментальной машине, а в целевой системе — так, например, устроена среда разработки для семейства микропроцессоров Transputer компании Intmos.

Системы виртуальных машин

Такие системы стоят несколько особняком.

Система виртуальных машин - это ОС, допускающая одновременную работу нескольких программ, но создающая при этом для каждой программы иллюзию того, что машина находится в полном ее распоряжении, как при работе под управлением отдельной ОС. Зачастую, "программой" оказывается полноценная операционная система - примерами таких систем являются VMWare и VirtualBox для машин с архитектурой x86, amd64 или VM для System/370 и ее потомков.

Виртуальные машины являются ценным средством при разработке и тестировании кросс-платформенных приложений. Реже они используются для отладки модулей ядра или самой операционной системы.

Такие системы отличаются высокими накладными расходами и сравнительно низкой надежностью, поэтому относительно редко находят промышленное применение. В системах виртуальных машин, как правило, приходится уделять много внимания эмуляции работы аппаратуры. Например, несколько программ могут начать программировать системный таймер. СВМ должна отследить такие попытки и создать для каждой из программ иллюзию, что она запрограммировала таймер именно так, как хотела. Разработка таких систем является сложным и часто неблагодарным делом. Архитектура таких систем сильно зависит от свойств виртуализируемой аппаратуры.

Архитектура мобильных ОС. Тенденции развития, особенности.

Лекция 5

Дизайн мобильных ОС прошел эволюцию от ОС для настольных рабочих станций через встраиваемые ОС до тех продуктов, которые мы видим в смартфонах сейчас.

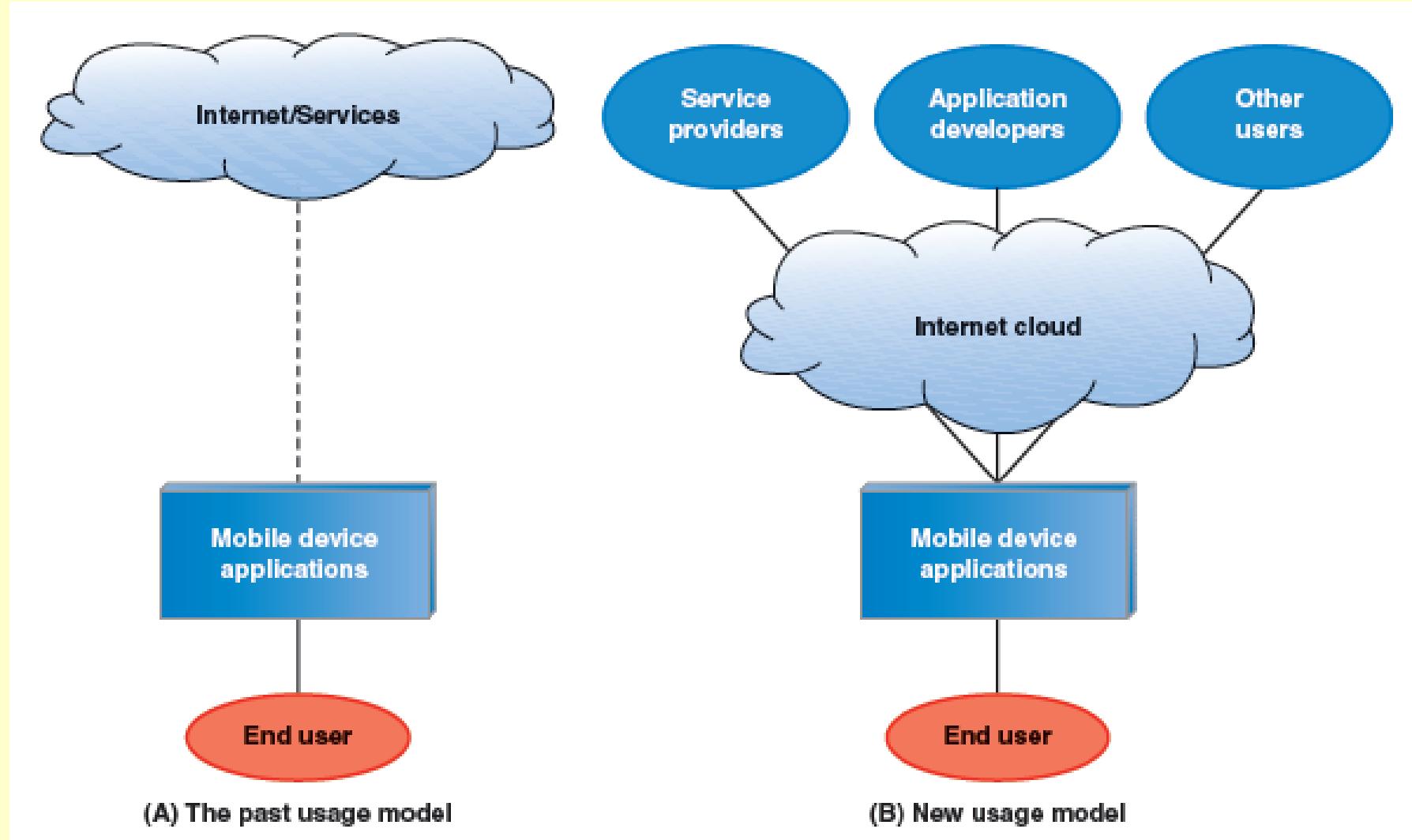
В течение этого процесса архитектура ОС менялась от сложной к простой и остановилась где-то на середине.

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 комфорта 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.

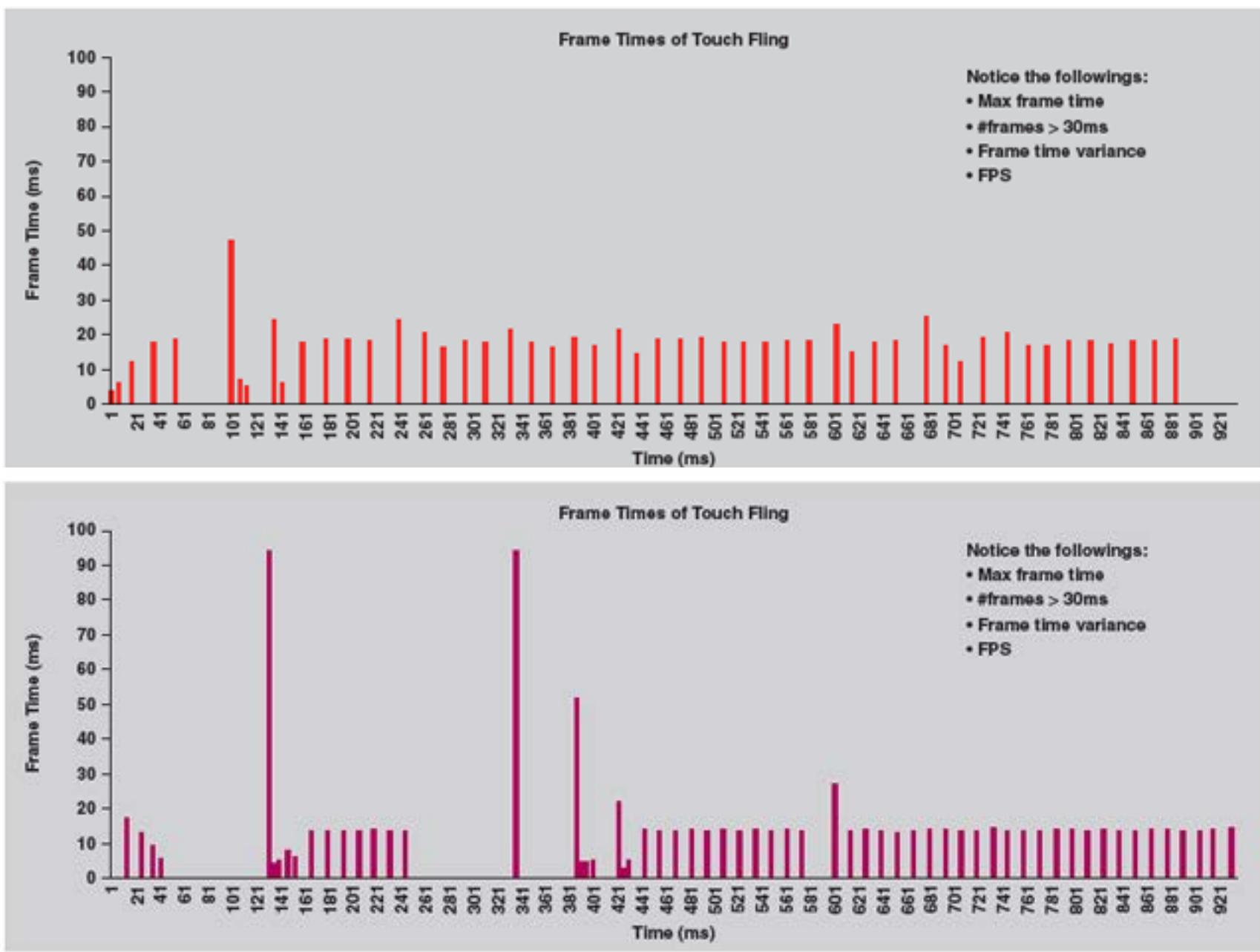
Другая проблема состоит в том, что FPS как ключевая величина гладкости взаимодействия не всегда отражает ощущения пользователя.

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.

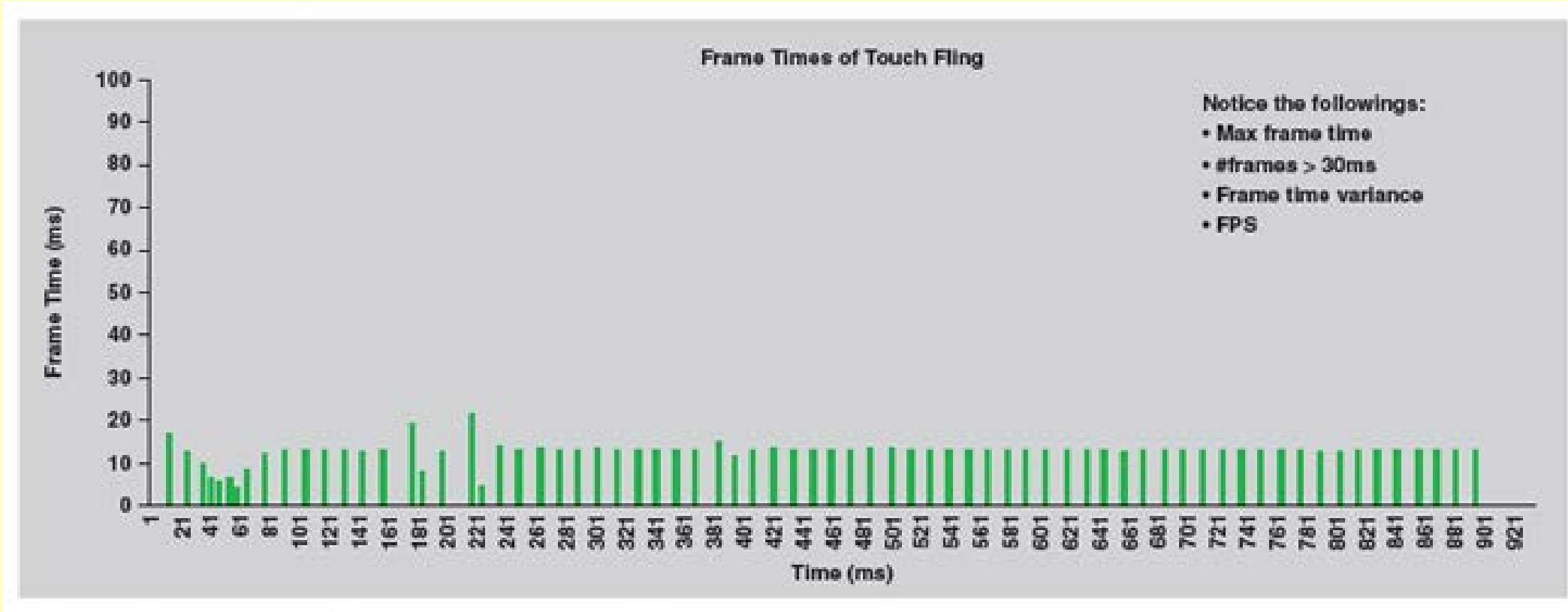
Теперь, наверное, причина видна всем: кроме FPS как такого, надо учитывать его стабильность, то есть вводить метрику максимального отклонения от средней величины.

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 нашем опыте разработки под Android, мы пришли 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 (*в статье приведены устаревшие данные, мы публикуем самые актуальные — прим. переводчика*).

Нам особенно приятно, что Tizen, новичок в мире мобильных ОС, занимает первое место в этом рейтинге.

Браузер	Платформа	Оценка + бонус
Tizen 2		492 + 16
BlackBerry 10	BlackBerry Q10 или Z10	485 + 11
Dolphin Engine	Android 2.2 или выше	469 + 3
Opera Mobile 14	Android	448 + 11
Tizen 1		426 + 16
Firefox Mobile 22	Различные платформы	422 + 14
Chrome 25	Android 4	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.

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_IDLE`, 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.

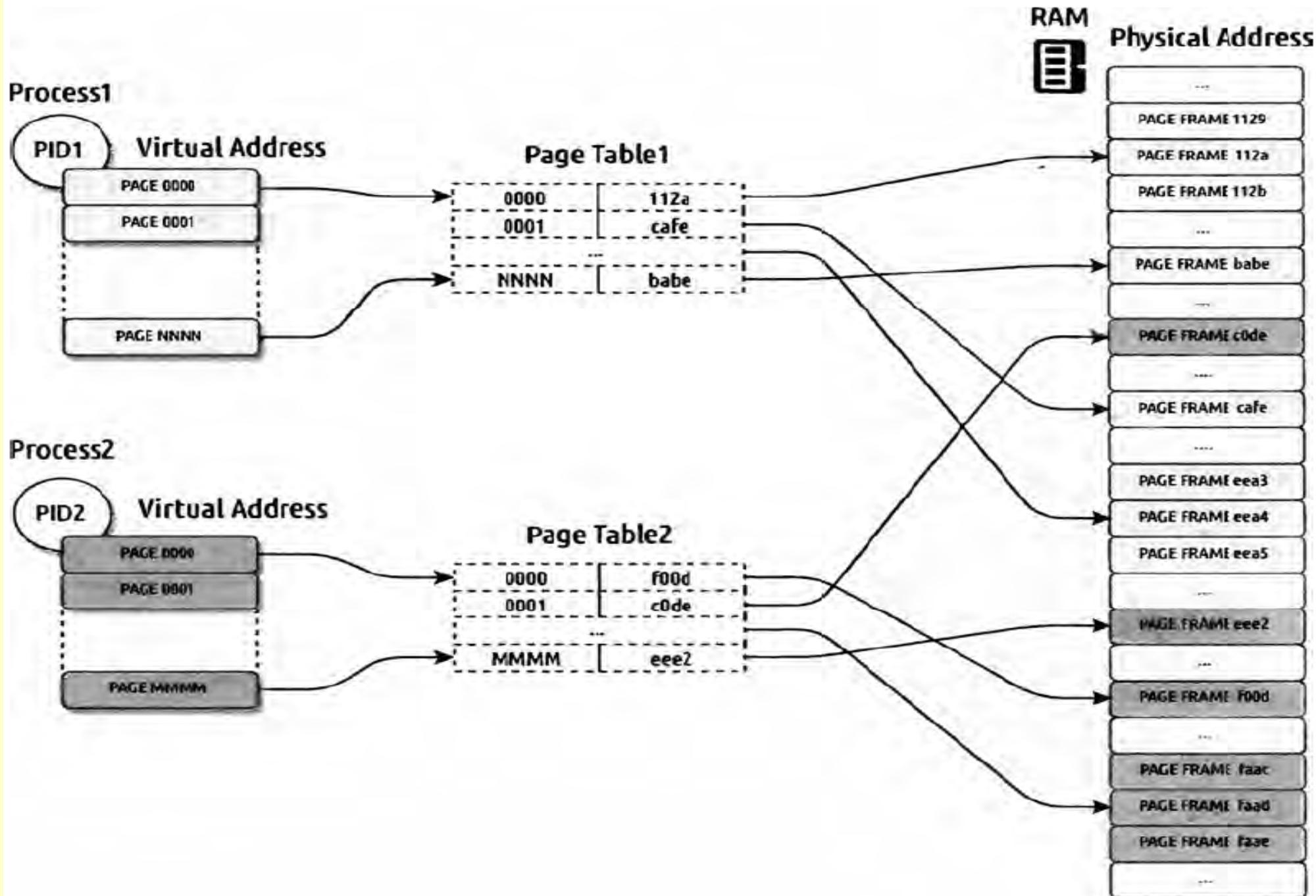


Рис. 4.3. Страницное отображение и распределение памяти

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.

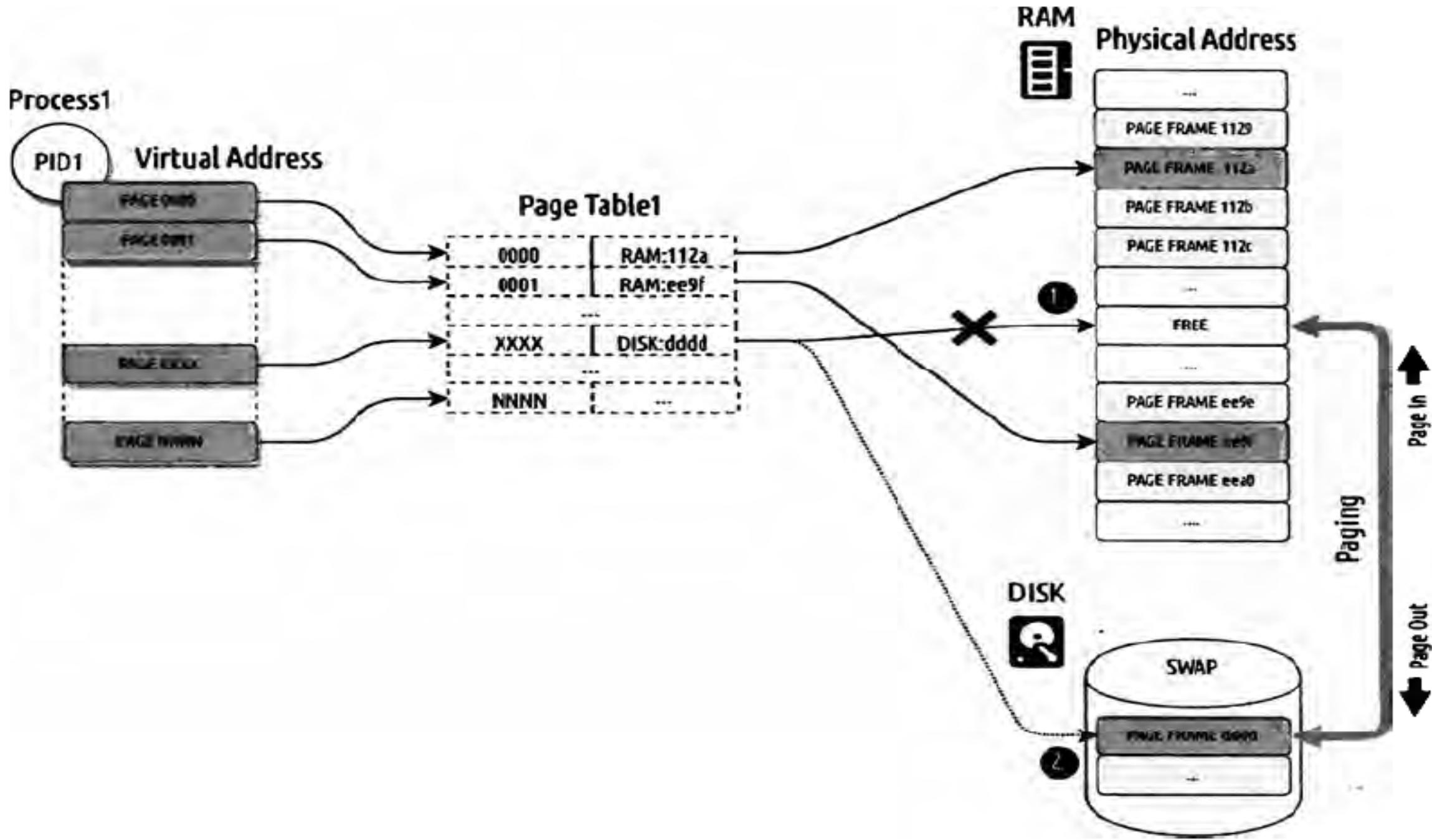


Рис. 4.4. Страницочный обмен

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.

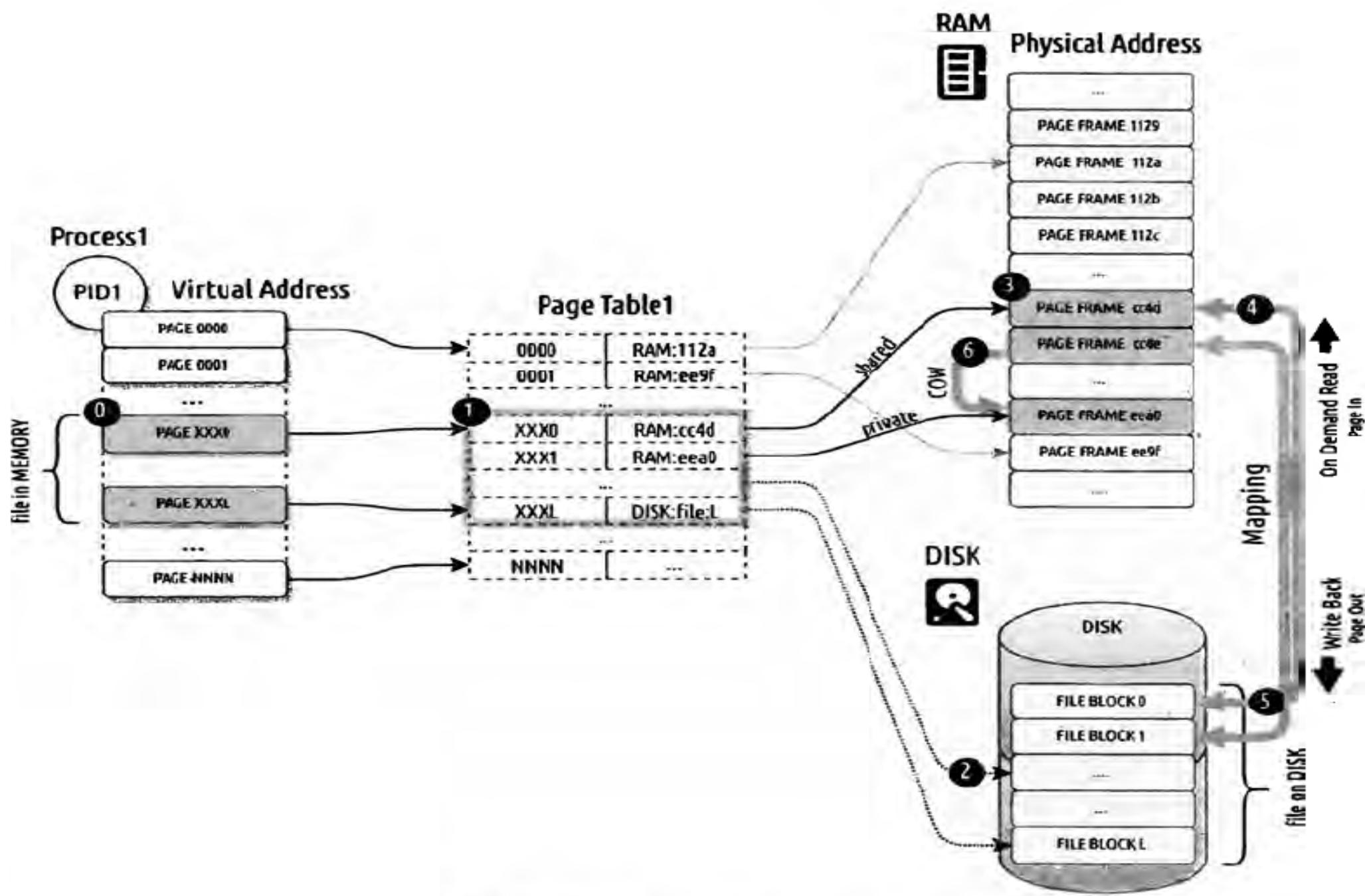


Рис. 4.5. Отображение файла в память

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	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

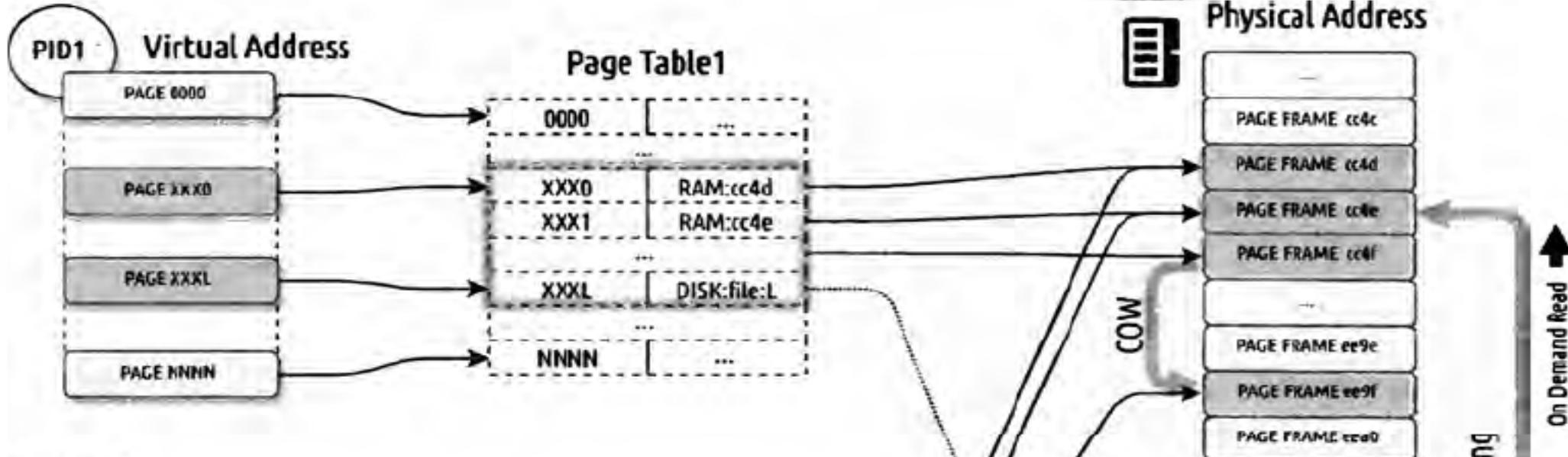
Механизм отображения считывает содержимое файла в страничные кадры только один раз, вне зависимости от количества процессов, отображающих этот файл в свою память.

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

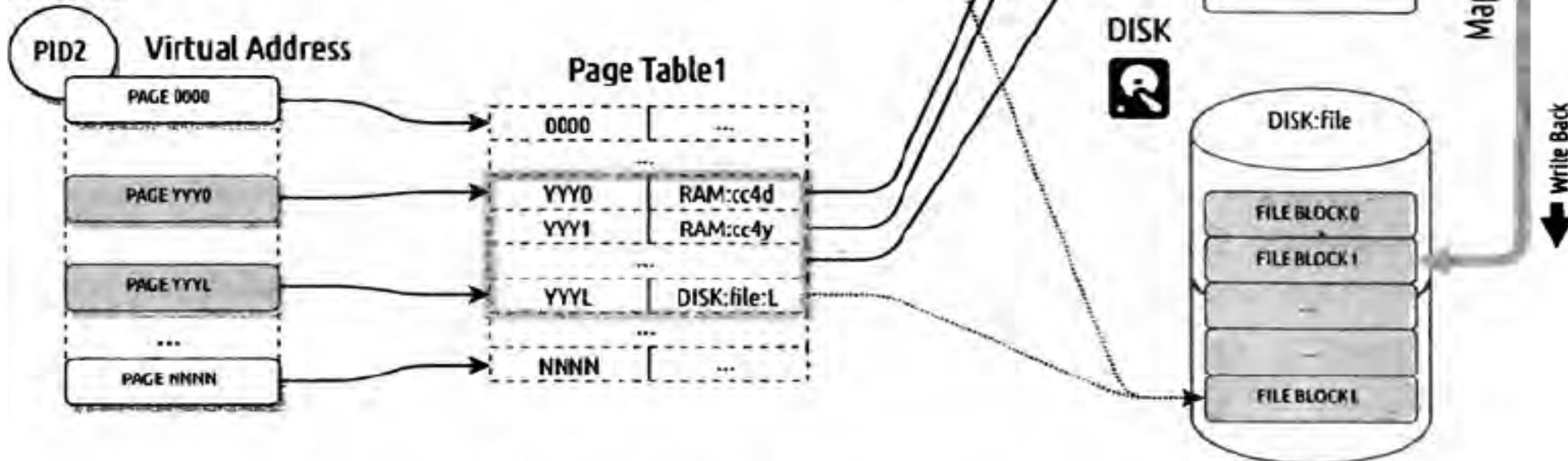


Рис. 4.6. Совместное использование памяти

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=4096 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 управляемых 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			

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 Добавка 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 выполняться 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.

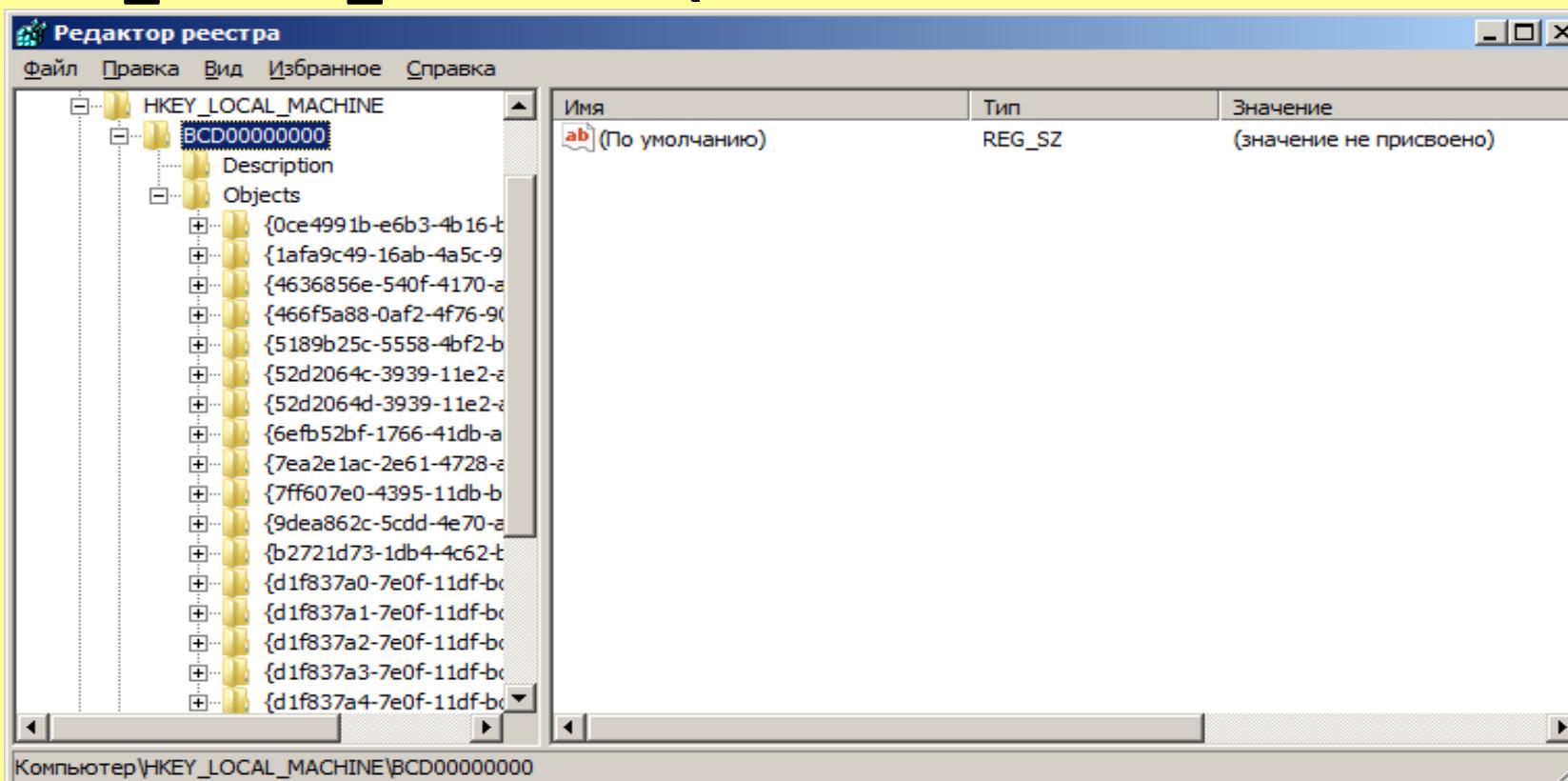
В Windows 7 разделы реестра SAM и SECURITY отображаются , однако не отображается их содержимое, для просмотра которого можно воспользоваться аналогичным приемом.

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.

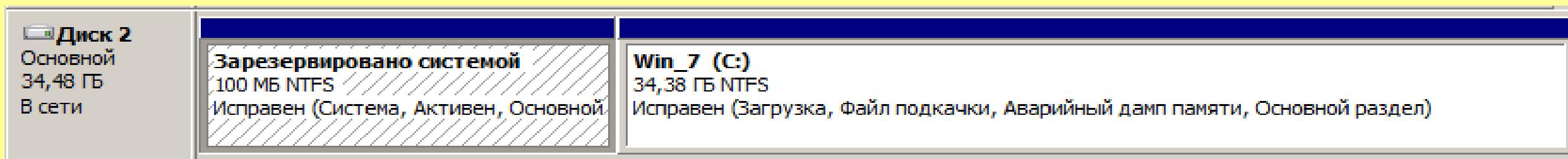
Нарушение целостности файлов реестра (нарушение структуры данных) или неверное значение отдельных критических параметров может привести к краху системы . Поэтому, прежде чем экспериментировать с реестром, позаботьтесь о возможности его сохранения и восстановления.

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.

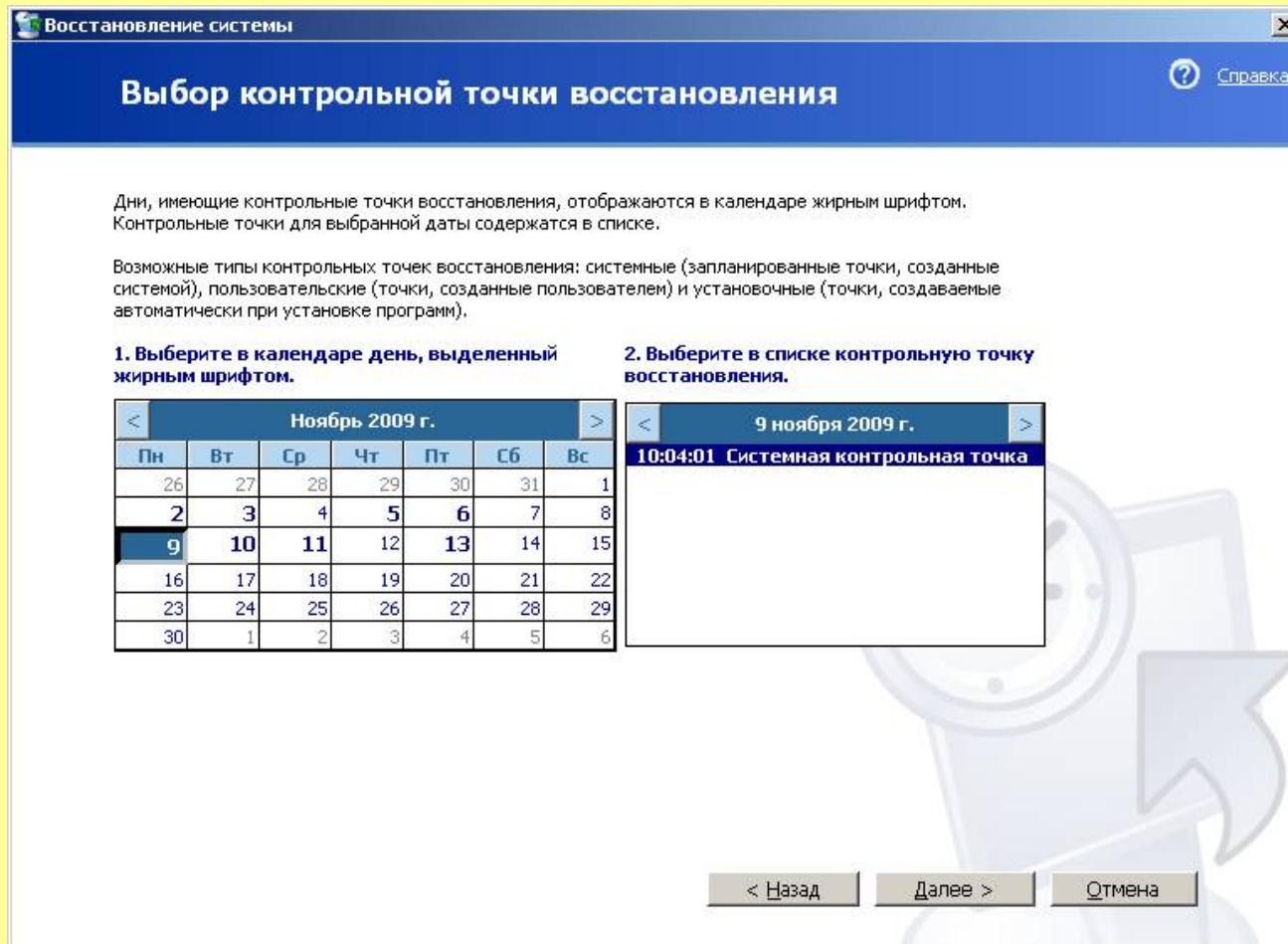
Сохранение и восстановление реестра

Использование точек восстановления (Restore Points)

В операционных системах Windows, начиная с Windows XP, существует механизм ,с помощью которого, при возникновении проблем, можно восстановить предыдущее состояние компьютера без потери личных файлов (документов Microsoft Word, рисунков, Избранного, Рабочего стола) с использованием **точек восстановления (Restore Points)**, которые при стандартных настройках, создаются системой автоматически во время простоя компьютера или во время существенных системных событий, таких, как установка нового приложения или драйвера. Кроме того, имеется возможность в любое время создать точку восстановления принудительно, с помощью "Панель управления" – "Система" – "Дополнительные параметры" – "Защита системы" – "Создать"

Точки восстановления представляют собой набор файлов операционной системы и реестра, скомпонованный по определенным правилам и сохраняемый в виде подкаталога в скрытой системной папке **System Volume Information**. Сохраненные в точке восстановления данные позволяют, практически гарантировано, вернуть операционную систему к состоянию на момент их создания. При изучении реестра (и отсутствии практического опыта работы с ним) принудительное создание точки восстановления перед началом работы позволит восстановить работоспособность Windows даже в случае краха системы. Стандартное средство восстановления системы работает только в среде самой Windows, однако существуют способы, которые позволяют вернуть систему к жизни даже при возникновении "синего экрана смерти" по причине неудачного редактирования реестра. Об этом чуть позже.

Для восстановления системы используется точками восстановления используется приложение "Восстановление системы" - **\windows\system32\restore\rstrui.exe** для Windows XP и **\windows\system32\rstrui.exe**



Можно также воспользоваться меню "Панель управления" – "Система" – "Дополнительные параметры" – "Задачи системы" – "Восстановление"

Данные точек восстановления хранятся в каталоге *System Volume Information* системного диска. Это скрытый системный каталог, доступ к которому разрешен только локальной системной учетной записи (*Local System*), или, другими словами, не пользователям, а "Службе восстановления системы". Поэтому, если вы хотите получить доступ к его содержимому, вам придется добавить права вашей учетной записи с использованием вкладки "Безопасность" в свойствах каталога "*System Volume Information*".

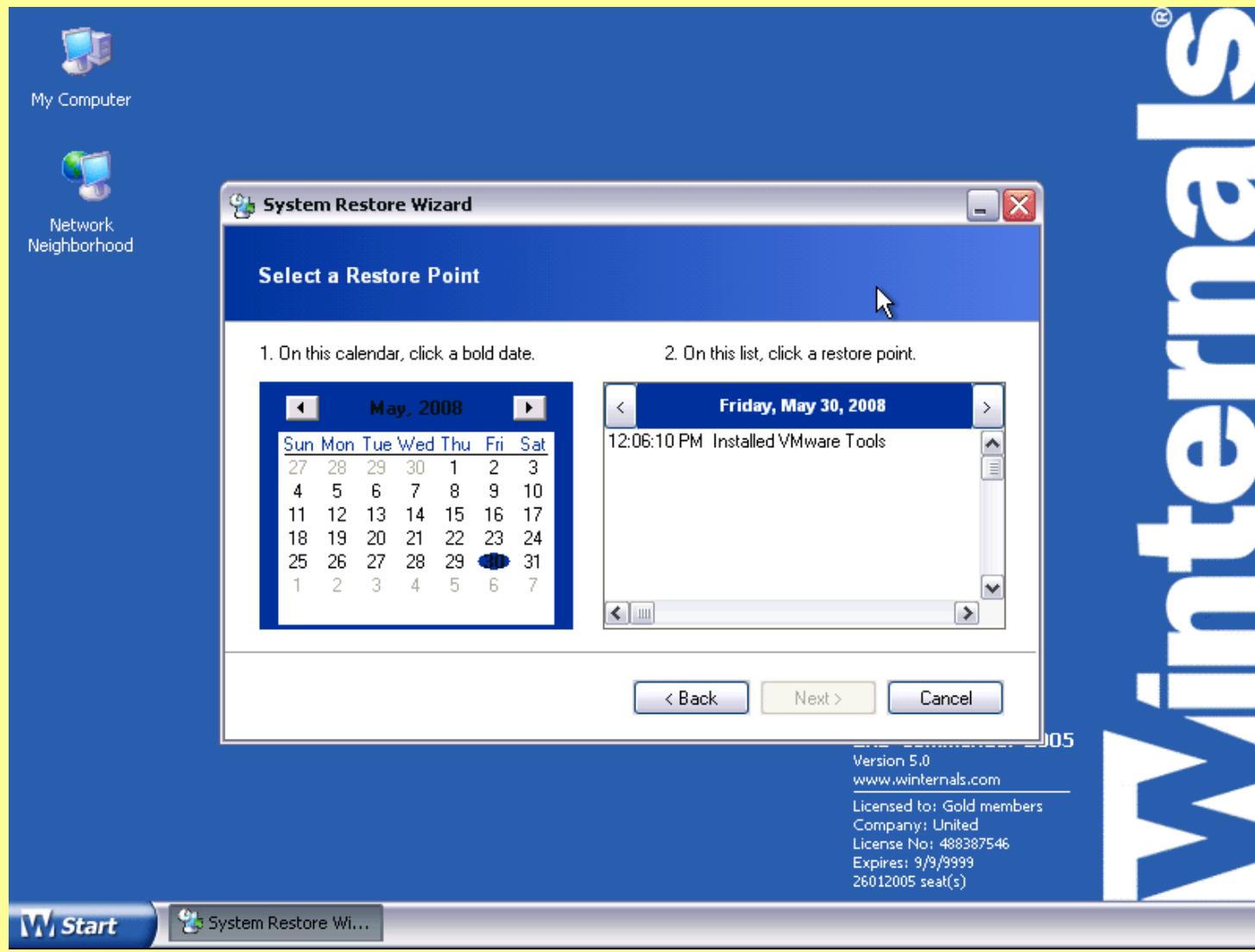
В случае Windows XP в папке **System Volume Information** есть подкаталог с именем, начинающимся с **_restore...** и внутри него - подкаталоги **RP0**, **RP1...**: - это и есть данные контрольных точек восстановления (Restore Point - RPn) для Windows XP. Внутри папок RPn имеется каталог с именем **snapshot** , содержащий копии файлов реестра на момент создания контрольной точки. При выполнении операции восстановления системы восстанавливаются основные системные файлы и файлы реестра. Соответственно, например, если крах системы вызван установкой нового драйвера, то после отката на точку восстановления, даже при наличии в каталоге Windows исполняемого файла драйвера, записи в реестре, обеспечивающие его загрузку, исчезнут, и система вернется в рабочее состояние. Подобным же образом можно избавиться от вирусного заражения, когда выполняется откат системы на тот момент, когда в реестре еще не было записей, обеспечивающих запуск вредоносного ПО.

Структура данных точек восстановления для Windows 7 отличается от той, что используется в Windows XP. Начиная с ОС Windows Vista, используется не только копирование файлов реестра и важнейших системных файлов, но и создаются снимки файловой системы (snapshot) службой теневого копирования томов. Снимки файловой системы также хранятся в каталоге **System Volume Information** и представляют собой сжатые файлы, в имени которых содержится глобальный идентификатор:

{3808876b-c176-4e48-b7ae-04046e6cc752}

Количество снимков зависит от настроек системы и наличия свободного места на диске. Однако порядок отката системы на состояние сохраненной точки восстановления такой же, как и для Windows XP.

Откат системы на точку восстановления является простым и эффективным способом восстановления работоспособности без потери пользовательских данных не только в случаях неудачного редактирования реестра, установки некорректно работающего системного программного обеспечения, но и, например, при вирусном заражении компьютера, когда имеется точка восстановления на момент времени, когда вируса еще не было в системе. Именно поэтому многие вредоносные программы пытаются уничтожить данные точки восстановления и отключить средства восстановления системы. В подавляющем большинстве случаев, запуск внедрившегося вируса обеспечивается определенными записями в реестре, а после отката эти записи исчезнут, и вирус не сможет получить управление, и тем самым, будет нейтрализован даже без использования антивирусного программного обеспечения. Как видно, механизм точек восстановления довольно эффективен, но воспользоваться им можно только в среде самой Windows - для выполнения отката неработоспособной системы, стандартно, нужно в ней же и загрузиться. А если система повреждена настолько, что загрузка невозможна, задача становится невыполнимой. Тем не менее, выход из данной ситуации (и даже не один) - есть. Можно, например, воспользоваться загрузочным диском Microsoft Diagnostic and Recovery Toolset (MS DaRT), он же (Winternals) ERD COMMANDER (ERDC), - инструментом на базе специальной версии Windows PE, загружаемой с компакт диска или другого внешнего носителя. ERD Commander умеет работать с точками восстановления той ОС Windows, которая была подключена к нему при загрузке, и позволяет легко выполнить откат на ее работоспособное состояние через меню "**Start - System Tools - System Restore**"

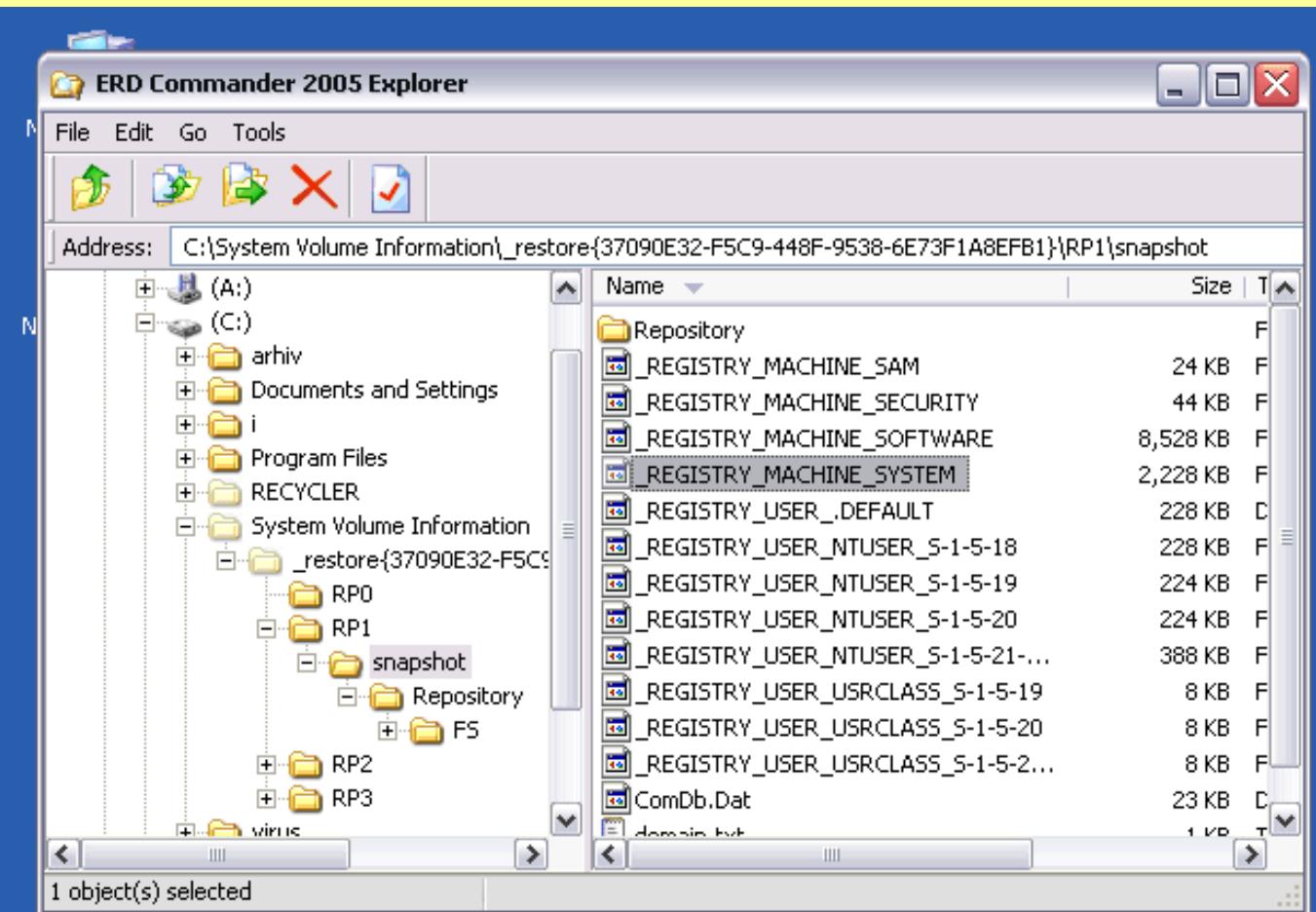


С помощью **System Restore Wizard** восстановление выполняется так же, как это можно было бы выполнить в среде рухнувшей системы.

Если же ERDC нет под рукой, или полный откат системы не нужен, можно воспользоваться ручным восстановлением отдельных файлов реестра, копии которых можно взять из данных точки восстановления для Windows XP, или из автоматически создаваемых копий файлов реестра в Windows 7. Для этого достаточно получить доступ к файловой системе жесткого диска с поврежденной Windows. Можно загрузиться в другой системе (Windows PE, Live CD, даже DOS с поддержкой файловых систем FAT32 / NTFS), получить доступ к данным системного диска проблемной Windows и просто заменить испорченный файл (ы) раздела реестра на файл (ы) из каталога точки восстановления .

Восстановление файлов реестра для Windows XP

В папке, где хранятся данные точек восстановления, **System Volume Information** на системном диске, находим подкаталог с именем, начинающимся с **_restore...** и внутри него - подкаталоги **RP0, RP1:** - это и есть искомые контрольные точки (Restore Point - RPx). Внутри папки RPx открываем каталог **SNAPSHOT**, содержащий копии файлов реестра, на момент создания контрольной точки.



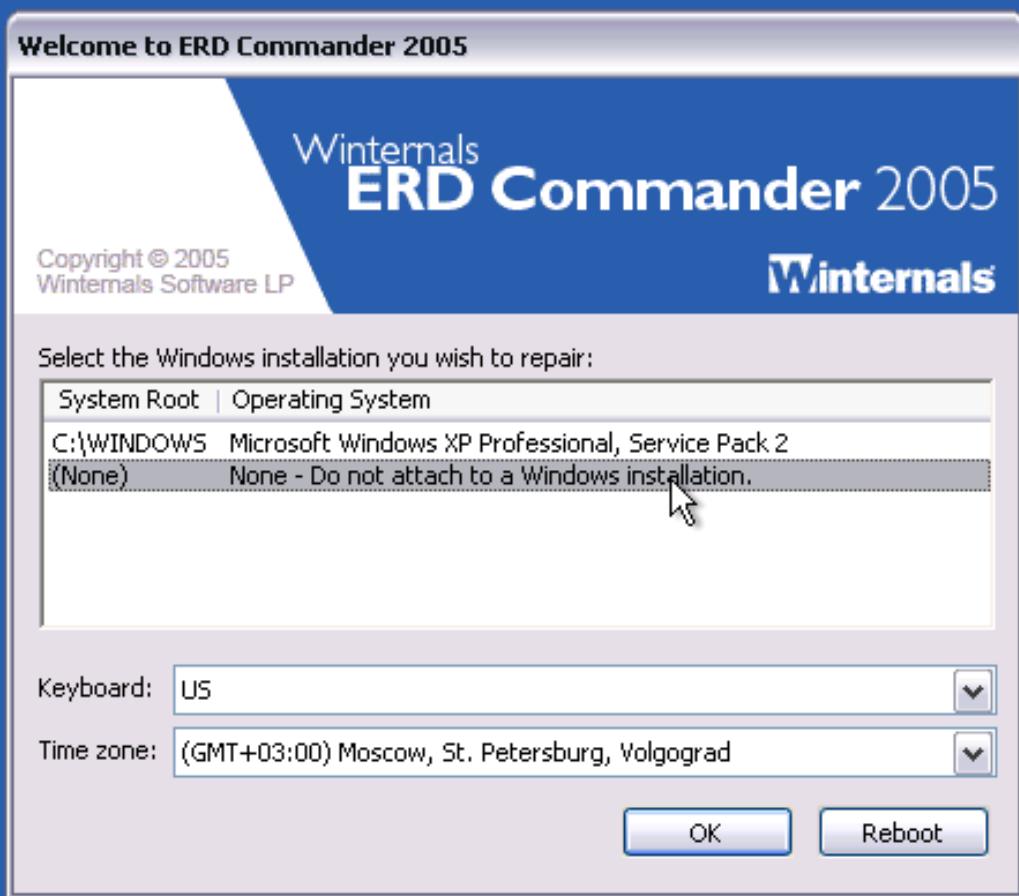
ERD Commander 2005
Version 5.0
www.winternals.com
Licensed to: Gold members
Company: United
License No: 488387546
Expires: 9/9/9999
26012005 seat(s)



Файл **REGISTRY_MACHINE_SYSTEM** - это и есть **копия файла SYSTEM**, он же - раздел реестра **HKEY_LOCAL_MACHINE\SYSTEM**. Остается лишь перетащить этот файл в каталог **\WINDOWS\SYSTEM32\CONFIG** и переименовать его. Запорченный файл **system** можно, на всякий случай, переименовать в **system.bad** или удалить.

Обычно, для восстановления работоспособности поврежденной операционной системы, достаточно копирования только файла **SYSTEM**, поскольку именно в нем хранятся наиболее важные параметры, необходимые для загрузки и функционирования ОС. Копирование файла **SOFTWARE** влияет на изменение состава установленного программного обеспечения для всех пользователей. При необходимости восстановления настроек пользователя нужно взять соответствующий ему файл **_REGISTRY_USER_NTUSER_S-1-5-21**: и скопировать его в каталог профиля (для Windows XP - обычно это "**:\Documents and Settings\Имя пользователя**") под именем **ntuser.dat**

Если при таком способе восстановления реестра будет использоваться **ERD Commander**, в режиме работы с выбранной Windows, то могут возникнуть проблемы с занятостью файлов. Чтобы этого не случилось, лучше в процессе загрузки не подключаться к проблемной операционной системе и выбрать **None**:



Восстановление файлов реестра для Windows 7 и более поздних ОС Windows

Для восстановления работоспособности Windows 7 и более поздних версий, удобнее всего воспользоваться средством Microsoft Diagnostic and Recovery Tools (MS DaRT), версии соответствующей версии и разрядности операционной системы. Для 32-разрядных и 64-разрядных ОС используются свои загрузочные диски. Наборы инструментов и их функциональные возможности, практически не отличаются от использовавшихся в ERD Commander, однако немного изменился их внешний вид и добавилась официальная поддержка русского языка. Описание работы с ERDC и MS DaRT 7.0 / 8.0 приводится в отдельной статье. В тех случаях, когда невозможно выполнить загрузку Windows, и запустить средство восстановления системы, инструменты MS DaRT позволяют выполнить ее откат на работоспособное состояние, практически так же, как и в случае с применением классического ERD Commander для Windows XP. Однако, описанный выше способ с частичной или полной заменой файлов реестра из данных точек восстановления применить не получится, поскольку возникает проблема извлечения из них отдельных кустов реестра. Тем не менее, в современных ОС семейства Windows существует более простая методика восстановления реестра, основанная на использовании автоматически создаваемых копий кустов (файлов реестра), создаваемых периодически специальной, задачей "Планировщика заданий". Задача **RegIdleBackup** создается в библиотеке планировщика заданий и выполняется скрытно, вне зависимости от регистрации пользователя, по умолчанию, один раз в 10 дней. Параметры задачи и сведения о ее выполнении можно посмотреть с помощью оснастки "Панель управления" - "Администрирование" - "Планировщик заданий". Открыть дерево "Библиотека планировщика" – Microsoft – Windows – Registry .

Планировщик заданий

Файл Действие Вид Справка

Левая панель: Планировщик заданий (Лог.)
Библиотека планировщика
Microsoft
Windows
Active Direct...
AppID
Application E...
Autochk
Bluetooth
CertificateSe...
Customer Ex...
Defrag
Diagnosis
DiskDiagnos...
Location
Maintenance
Media Center
MemoryDiagr...
MobilePC
MUI
Multimedia
NetTrace

Средняя панель: Таблица заданий

Файл	Состояние	Триггеры	Время следующего запуска	Время прошлого запуска
RegIdleBackup	Готово	В 0:00 каждые 10 дн.	16.10.2014 0:37:55	06.10.2014

Правая панель: Действия

Registry

- Создать...
- Создать...
- Изменить...
- Открыть...
- Включить...
- Создать...
- Удалить...
- Вид
- Обзор...
- Справка...

Выбрано:

- Выполнить...
- Задача...

Хотя задано время выполнения 00:00, в параметрах задачи установлен режим немедленного запуска, если просрочен плановый запуск. Таким образом, задача будет выполнена, даже если компьютер на момент планового запуска был выключен, и в случае успешного завершения, каталоге **\windows\system32\config\RegBack** будет создана резервная копия файлов реестра default, sam, security, software и system. Следовательно, в тех случаях, когда восстановление системы стандартными средствами невозможно (например, нет данных точек восстановления), можно воспользоваться ручным копированием файлов реестра из каталога **windows\system32\config\RegBack** в каталог **windows\system32\config** так же, как и для случая описанного выше.

Кроме того, как уже упоминалось выше, в операционных системах Windows Vista и старше, при создании точки восстановления системы, создается и теневая копия тома, представляющая собой полный снимок файловой системы (snapshot, снапшот), который тоже может помочь в восстановлении не только файлов реестра, но и любых других, существовавших на момент создания снимка. Для получения информации о существующих теневых копиях файловой системы можно воспользоваться командой:

vssadmin list shadows

Для доступа к данным теневых копий можно использовать , например, создание символической ссылки на корневой каталог теневой копии тома, командой:

mklink /D C:\shadow1 \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1

Командная строка должна быть запущена от имени администратора. После выполнения команды, на диске С: появится папка **shadow1** , содержимое которой представляет собой копию диска С: на момент создания точки восстановления.

Для доступа к файлам и папкам из снимка файловой системы, можно использовать и сторонние программы, умеющие работать с томами теневых копий, как, например, популярная программа восстановления данных **Recuva**.

Использование утилиты для работы с реестром из командной строки REG.EXE

В ОС Windows 2000 утилита REG.EXE входила в состав пакета Support tools, в Windows XP , windows 7 /8 - входит в стандартный набор программ, устанавливаемых в процессе инсталляции системы. На практике, можно для Windows 2000 использовать REG.EXE из комплекта Windows XP - просто скопируйте ее в каталог \winnt\system32). Запускается из командной строки. При запуске без параметров выдает краткую справку по использованию:

**Программа редактирования системного реестра из командной строки, версия 3.0
(C) Корпорация Майкрософт, 1981-2001. Все права защищены**

REG <Операция> [Список параметров]

**<Операция> == [QUERY | ADD | DELETE | COPY |
SAVE | LOAD | UNLOAD | RESTORE |
COMPARE | EXPORT | IMPORT]**

Код возврата: (за исключением REG COMPARE)

0 - Успешно

1 - С ошибкой

Для получения справки по определенной операции введите:

REG /?

Примеры:

REG QUERY /?

REG ADD /?

REG DELETE /?

REG COPY /?

REG SAVE /?

REG RESTORE /?

REG LOAD /?

REG UNLOAD /?

REG COMPARE /?

REG EXPORT /?

REG IMPORT /?

Для резервного копирования реестра используется REG.EXE SAVE, для восстановления - REG.EXE RESTORE

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. Есть, правда некоторые моменты. В версии утилиты из ОС Windows 2000 нельзя было указывать путь в имени файла для сохранения раздела реестра и сохранение выполнялось только в текущий каталог. 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, 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 – 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. И если в Windows 7 существует задание планировщика для копирования файлов реестра, созданное при установке системы, то в Windows XP, такого задания нет, и, при нарушении целостности реестра, шансов на восстановление становится меньше. Что бы исключить подобную ситуацию, было бы неплохо, позаботиться об автоматическом резервированияния файлов реестра без участия человека. Например, с помощью командного файла, выполняемого планировщиком или в процессе регистрации пользователя.

Для создания полной копии реестра Windows XP в командной строке удобно использовать бесплатную консольную утилиту **regsaver.exe** - 380кб

Утилита сохраняет файлы реестра в каталог, указываемый в качестве параметра командной строки:

regsaver.exe D:\regbackup

После выполнения программы в каталоге D:\regbackup будет создан подкаталог с уникальным именем, состоящим из года, месяца, числа и времени создания резервной копии файлов реестра ("ууууттмддhhmmss"). После выполнения резервирования программа может выключить компьютер или перевести его в спящий режим:

regsaver.exe D:\regbackup /off /ask - Выключить компьютер. Ключ /ask требует подтверждения пользователя на выполнение выключения питания.

regsaver.exe D:\regbackup /standby - Перевести в спящий режим без подтверждения (нет /ask)

regsaver.exe D:\regbackup /hibernate /ask - Перевести в режим Hibernate

Вместо стандартного выключения компьютера можно использовать резервное копирование реестра с выключением по его завершению.

Важным преимуществом консольной версии является то, что можно создать командный файл и выполнять его с помощью планировщика заданий **для автоматического создания** резервных копий файлов реестра.

Я неоднократно использовал `regsaver` в сценариях регистрации пользователей при входе в домен для периодического (например, 1 раз в неделю) копирования файлов реестра, и наличие резервной копии нередко выручало в критической ситуации. Для периодического создания резервных копий файлов реестра можно воспользоваться запуском утилиты от имени администратора с сохранением полномочий.

`runas /savecred administrator "regsaver.exe D:\regbackup"`

Параметр `/savecred` используется для запоминания полномочий (credentials) пользователя с именем `administrator` при выполнении от его имени задания `regsaver.exe D:\regbackup`. При первом запуске `runas`, будет выдан запрос на ввод пароля указанного пользователя, который будет запомнен и не будет запрашиваться при последующих запусках.

Утилита **regsaver** выполняется без ошибок в среде Windows 7, однако, копии файлов реестра будут не полными, и использовать их в полной мере невозможно, да и нет необходимости, поскольку автоматическое резервирование файлов реестра выполняется заданием RegIdleBackup, автоматически выполняемым "Планировщиком заданий" Windows. Копии файлов реестра Windows 7/8 хранятся в каталоге C:\Windows\System32\config\RegBack\. Состояние задачи планировщика, в том числе, запланированное время выполнения резервного копирования файлов реестра можно посмотреть при выполнении команды

```
schtasks /query /tn \Microsoft\Windows\Registry\RegIdleBackup
```

Пример отображаемой информации:

Папка: \Microsoft\Windows\Registry

Имя задачи	Время следующего запуска	Состояние
===== =====	=====	=====
RegIdleBackup	23.02.2015 0:52:15	Готово

Восстановление реестра, при отсутствии резервных копий в Windows XP.

Иметь актуальные резервные копии реестра - это практически, всегда выход из ситуации когда работа Windows завершается критической ошибкой или, например, при загрузке системы, вы видите сообщение о нарушении целостности куста реестра SYSTEM:

**Windows XP could not start because the following file is missing or corrupt:
\WINDOWS\SYSTEM32\CONFIG\SYSTEM**

Если есть резервная копия, выполняем за 5-10 минут восстановление файла SYSTEM, как описывалось выше, и система продолжает работать, как ни в чем не бывало. Я, конечно, не рассматриваю здесь случай, когда некондиционность файла реестра вызвана сбоями оборудования компьютера.

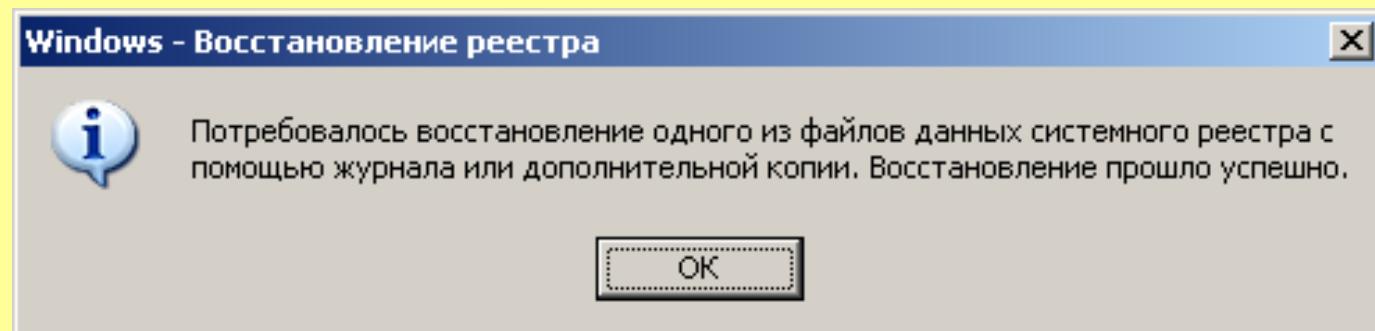
Если же, резервирование данных реестра никогда не выполнялось, был отключен механизм создания контрольных точек восстановления, или вы использовали Win2K, где этого механизма просто отсутствует, то все равно, некоторые шансы оживить систему, есть.

Возможно, поможет восстановить систему:

- использование резервных файлов реестра, автоматически созданных каким-либо программным обеспечением. Откройте папку \Windows\system32\config и проверьте, нет ли в ней файла **system.bak** (возможно другое расширение, отличное от .alt и .log). Поврежденный файл system переименуйте или сохраните в каком-либо ином месте). Найденный файл- копию system... переименуйте в system и попробуйте загрузиться.
- в Windows XP возможно использование, сохраненного после начальной установки, файла (файлов) из каталога \WINDOWS\REPAIR. Такой вариант, не самый оптимальный, на крайний случай. После завершения установки Windows, копии файлов реестра сохраняются в упомянутом каталоге и файл system будет соответствовать состоянию ОС сразу после завершения установки.
- использование **функции восстановления** редактора реестра Windows XP при загрузке поврежденного куста.

Редактор реестра позволяет открывать файлы не только "своего" реестра, но и файлы, являющиеся реестром другой операционной системы. И имеется возможность восстановления поврежденного куста при загрузке в редакторе реестра. Для этого

- Загрузитесь в другой операционной системе Windows с возможностью загрузки проблемного куста реестра .
 - Запустите редактор реестра.
 - В левой части дерева реестра выберите один из разделов: HKEY_USERS или HKEY_LOCAL_MACHINE.
 - В меню **Реестр (Registry)** (В других версиях редактора реестра этот пункт меню может называться "Файл") выберите команду "**Загрузить куст(Load Hive)**".
 - Найдите испорченный куст (в нашем случае - system).
 - Нажмите кнопку **Открыть**.
 - В поле **Раздел** введите имя, которое будет присвоено загружаемому кусту. Например BadSystem.
- После нажатия **OK** появится сообщение:



В левом окне редактора реестра выберите подключенный куст (BadSystem) и выполните команду "**Выгрузить куст**". Поврежденный system будет восстановлен. Причем, редактор реестра Windows XP вполне успешно восстановит реестр и более старой ОС Windows 2000. Даже если содержимое восстановленного таким образом файла будет не совсем актуальным, система, с большой долей вероятности, останется работоспособной. Возможно, придется переустановить некоторые программные продукты, или обновить драйверы.

Мониторинг реестра. Как определить, какие программы обращаются к реестру.

Одной из лучших программ для мониторинга реестра, является утилита **Process Monitor**, которую можно загрузить с сайта Microsoft, по ссылке на странице программы:

[Страница Process Monitor на Microsoft Sysinternals](#)

Программа Process Monitor является усовершенствованным инструментом отслеживания активности приложений и системных служб , который в режиме реального времени отображает активность файловой системы, реестра, сети, процессов и потоков. В этой программе сочетаются возможности двух ранее выпущенных программ от Sysinternals: Filemon и Regmon, а также огромный ряд улучшений, включая расширенную фильтрацию, всеобъемлющие свойства событий, такие как ID сессий и имена пользователей, достоверную информацию о процессах, полноценный стек потока со встроенной поддержкой всех операций, одновременную запись информации в файл и многие другие возможности. Эти уникальные возможности делают программу Process Monitor ключевым инструментом для устранения неполадок и избавления от вредоносных программ. Недостатком программы можно считать вероятность зависания при большом количестве отслеживаемых событий.

Не смотря на появление **Process Monitor**, в среде Windows XP удобнее использовать хотя и устаревшую, но по прежнему эффективную, утилиту **Regmon**, поддержка которой прекращена автором.

Фактически, **Regmon**, является частью утилиты **Process Monitor**, и выполняет только мониторинг обращений к реестру. Параметры программы, панель инструментов, принципы фильтрации и выделения отслеживаемых событий, а также результатов мониторинга в обеих программах практически одинаковы.

Информация выдается в удобном виде, который можно настроить под свои нужды - исключить из результатов мониторинга данные о работе с реестром неинтересных вам приложений, подчеркнуть выбранным цветом то, что считаете особо важным, включить в результаты мониторинга только выбранные процессы. Программа позволяет быстро и легко выполнить запуск редактора реестра с переходом к указанному разделу или параметру. Имеется возможность выполнять мониторинг в процессе загрузки операционной системы с записью результатов в специальный журнал %SystemRoot\Regmon.log.

После старта RegMon, можно определить критерии фильтрации результатов мониторинга реестра:

Regmon Filter

X

These filters were active the last time you exited Regmon. The settings you configure will be enabled when Regmon starts.

OK

Enter multiple filter match strings separated by the ';' character. '*' is a wildcard.

Cancel

Apply

Include:

*



Exclude:



Highlight:



Log Opens:



Log Reads:



Log Successes:



Log Writes:



Log Errors:



Defaults

По умолчанию протоколируются все события обращения к реестру. Фильтр задается значениями полей:

Include - Если * - выполнять мониторинг для всех процессов. Имена процессов разделяются символом ";". Например -**FAR.EXE;Winlogon.exe** - будут фиксироваться обращения к реестру только для процессов far.exe и winlogon.exe.

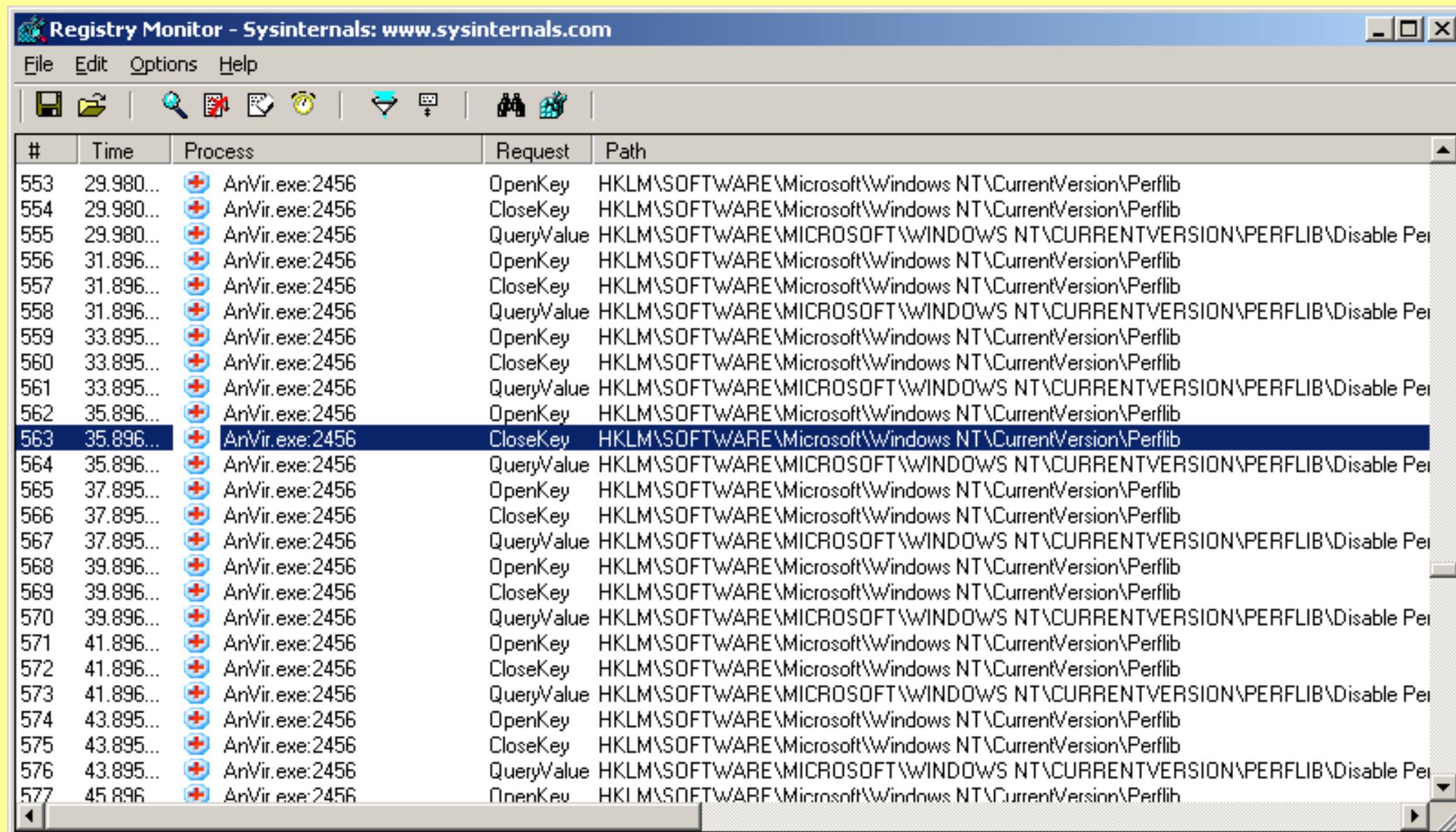
Exclude - какие процессы исключить из результатов мониторинга.

Highlight - какие процессы выделить выбранным цветом (по умолчанию - красным).

Значения полей фильтра запоминаются и выдаются при следующем старте Regmon. При нажатии кнопки **Defaults** выполняется сброс фильтра в установки по умолчанию - фиксировать все обращения к реестру. Значения полей фильтра удобнее формировать не при старте RegMon, а в процессе мониторинга, используя меню правой кнопки мыши для выбранного процесса

- **Include process** - включить данный процесс в мониторинг
- **Exclude process** - исключить данный процесс из мониторинга.

После старта Regmon с фильтрами по умолчанию, вы увидите большое количество записей об обращении к реестру и, используя Include/Exclude process, можете настроить вывод результатов только нужного вам процесса (процессов).



The screenshot shows the Registry Monitor application window. The title bar reads "Registry Monitor - Sysinternals: www.sysinternals.com". The menu bar includes "File", "Edit", "Options", and "Help". Below the menu is a toolbar with icons for file operations like Open, Save, Find, and Filter. The main area is a table with columns: #, Time, Process, Request, and Path. The table lists 577 rows of registry activity. Most entries show requests from the "AnVir.exe:2456" process. The row at index 563 is highlighted with a blue background, indicating it is selected. The "Request" column shows actions like OpenKey, CloseKey, and QueryValue, and the "Path" column shows registry keys under HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib and HKLM\SOFTWARE\MICROSOFT\WINDOWS NT\CURRENTVERSION\PERFLIB\Disable Per.

#	Time	Process	Request	Path
553	29.980...	AnVir.exe:2456	OpenKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib
554	29.980...	AnVir.exe:2456	CloseKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib
555	29.980...	AnVir.exe:2456	QueryValue	HKLM\SOFTWARE\MICROSOFT\WINDOWS NT\CURRENTVERSION\PERFLIB\Disable Per
556	31.896...	AnVir.exe:2456	OpenKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib
557	31.896...	AnVir.exe:2456	CloseKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib
558	31.896...	AnVir.exe:2456	QueryValue	HKLM\SOFTWARE\MICROSOFT\WINDOWS NT\CURRENTVERSION\PERFLIB\Disable Per
559	33.895...	AnVir.exe:2456	OpenKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib
560	33.895...	AnVir.exe:2456	CloseKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib
561	33.895...	AnVir.exe:2456	QueryValue	HKLM\SOFTWARE\MICROSOFT\WINDOWS NT\CURRENTVERSION\PERFLIB\Disable Per
562	35.896...	AnVir.exe:2456	OpenKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib
563	35.896...	AnVir.exe:2456	CloseKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib
564	35.896...	AnVir.exe:2456	QueryValue	HKLM\SOFTWARE\MICROSOFT\WINDOWS NT\CURRENTVERSION\PERFLIB\Disable Per
565	37.895...	AnVir.exe:2456	OpenKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib
566	37.895...	AnVir.exe:2456	CloseKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib
567	37.895...	AnVir.exe:2456	QueryValue	HKLM\SOFTWARE\MICROSOFT\WINDOWS NT\CURRENTVERSION\PERFLIB\Disable Per
568	39.896...	AnVir.exe:2456	OpenKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib
569	39.896...	AnVir.exe:2456	CloseKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib
570	39.896...	AnVir.exe:2456	QueryValue	HKLM\SOFTWARE\MICROSOFT\WINDOWS NT\CURRENTVERSION\PERFLIB\Disable Per
571	41.896...	AnVir.exe:2456	OpenKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib
572	41.896...	AnVir.exe:2456	CloseKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib
573	41.896...	AnVir.exe:2456	QueryValue	HKLM\SOFTWARE\MICROSOFT\WINDOWS NT\CURRENTVERSION\PERFLIB\Disable Per
574	43.895...	AnVir.exe:2456	OpenKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib
575	43.895...	AnVir.exe:2456	CloseKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib
576	43.895...	AnVir.exe:2456	QueryValue	HKLM\SOFTWARE\MICROSOFT\WINDOWS NT\CURRENTVERSION\PERFLIB\Disable Per
577	45.896	AnVir.exe:2456	OpenKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib

Назначение колонок:

- номер по порядку

Time - Время. Формат времени можно изменить с помощью вкладки **Options**

Process - имя процесса: идентификатор процесса (PID)

Request - тип запроса. OpenKey - открытие ключа (подраздела) реестра, CloseKey - закрытие, CreateKey - создание, QueryKey - проверка наличия ключа и получение количества вложенных ключей (подразделов, subkeys),

EnumerateKey - получить список имен подразделов указанного раздела,

QueryValue - прочитать значение параметра, SetValue - записать значение.

Path - путь в реестре.

Result - результат выполнения операции. SUCCESS - успешно, NOT FOUND - ключ (параметр) не найден. ACCESS DENIED - доступ запрещен (недостаточно прав). Иногда бывает BUFFER OVERFLOW - переполнение буфера - результат операции не помещается в буфере программы.

Other - дополнительная информация, результат выполненного запроса.

Программа очень проста в использовании. После старта, лучше выбрать фильтр по умолчанию, т.е. фиксировать все обращения к реестру, а затем, в основном окне программы, выбрать ненужный процесс и с помощью правой кнопки мыши вызвать контекстное меню - Exclude process - информация об обращении к реестру данного процесса выводиться не будет. И таким же образом отфильтровать другие, не интересующие вас процессы. Небольшим недостатком программы является ограниченное количество возможных фильтров.

При работе с программой можно использовать меню File, Edit, Options или сочетание клавиш:

CTRL-S - сохранить результаты

CTRL-P - свойства выбранного процесса

CTRL-E - включить/выключить мониторинг

CTRL-F - поиск по контексту

CTRL-C - копировать выбранную строку в буфер обмена

CTRL-T - изменить формат времени

CTRL-X - очистить окно результатов мониторинга

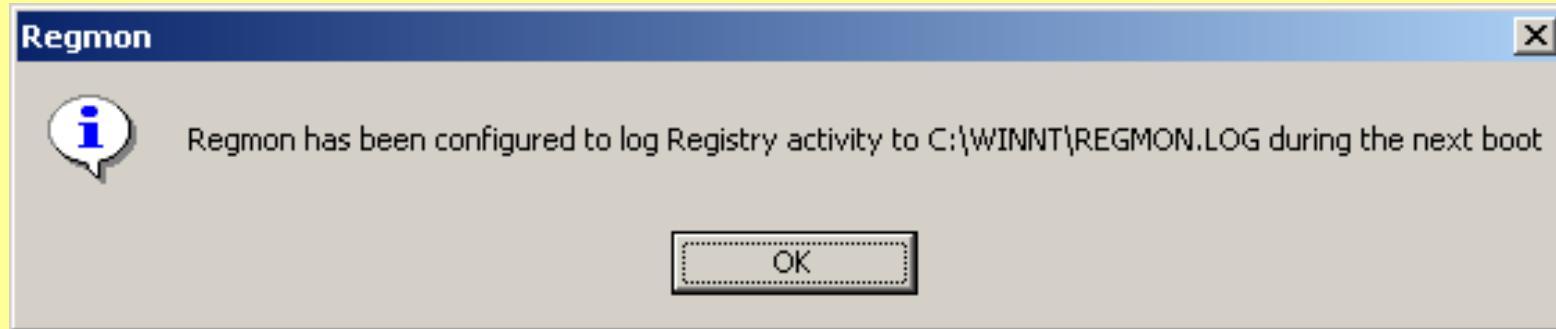
CTRL-J - запустить редактор реестра и открыть ветвь, указанную в колонке Path. Это же действие выполняется при двойном щелчке левой кнопки мыши. Очень полезная возможность, позволяет значительно экономить время.

CTRL-A - включить/выключить автоматическую прокрутку

CTRL-H - позволяет задать число строк результатов мониторинга

Еще одна очень полезная возможность - получить журнал обращений к реестру в процессе загрузки операционной системы.

Для этого выбираете меню **Options-Log Boot**. Программа выдаст сообщение, что Regmon сконфигурирован для записи обращений к реестру в файл журнала в ходе следующей перезагрузки ОС:



После перезагрузки ОС, в корневом каталоге системы (C:\Windows) будет находиться файл Regmon.log с журналом результатов мониторинга. Режим записи в журнал будет продолжаться до запуска Regmon.exe вошедшем в систему пользователем и выполняется только для одной перезагрузки системы. Конечно же, содержимое журнала не будет полностью отображать абсолютно все обращения к реестру. Поскольку Regmon в режиме Log Boot инсталлируется в системе и, после перезагрузки, запускается в качестве драйвера, все обращения к реестру, произошедшие до его старта, в журнале не зафиксируются. Однако большая часть все же туда попадет, и вы увидите, что таких обращений будет несколько сотен тысяч.

Работа с конфигурационными файлами в Linux

Основная цель данной лекции - изучить основы работы с системными конфигурационными файлами ОС Linux для автоматизации процессов системного администрирования в операционной системе Linux.

- /etc/adjtime
- /etc/bash.bashrc
- /etc/crontab
- /etc/environment
- /etc/fstab
- /etc/group
- /etc/hostname
- /etc/hosts
- /etc/hosts.allow и /etc/hosts.deny
- /etc/issue и /etc/issue.net
- /etc/ld.so.conf
- /etc/localtime
- /etc/login.defs
- /etc/mime.types
- /etc/modprobe.d/
- /etc/modules-load.d/
- /etc/nsswitch.conf
- /etc/ntp.conf
- /etc/os-release
- /etc/passwd
- /etc/profile
- /etc/resolv.conf
- /etc/sddm.conf
- /etc/shadow
- /etc/sudoers
- /etc/sysctl.conf
- /etc/vconsole.conf
- /boot/grub/grub.cfg

Общие сведения о конфигурационных файлах в Linux

Операционная система Linux в отличие от Windows не имеет общего реестра для хранения настроек системы, все настройки хранятся в конфигурационных файлах.

Большинство этих файлов размещено в папке `/etc/`.

Настройки большинства системных и сторонних программ находятся в этих файлах, это могут быть настройки графического сервера, менеджера входа, системных служб, веб-сервера, системы инициализации.

Только часть файлов конфигурации находятся в других папках, например, файлы настройки рабочего окружения в домашнем каталоге пользователя.

Очень важно понимать, за что отвечают те или иные конфигурационные файлы, чтобы при необходимости очень быстро сориентироваться. В этой лекции мы рассмотрим основные конфигурационные файлы Linux, их расположение и предназначение.

Большинство файлов, которые мы привыкли считать стандартными, относятся к системе инициализации или к другим системным утилитам. В основном эти файлы размещены в папке /etc.

Название этой папки расшифровывается как "et cetera", что с латинского означает "и другие" или "и так далее". Сначала давайте посмотрим содержимое каталога /etc Linux:

`ls -l /etc/`

Здесь достаточно много различных файлов. Дальше мы рассмотрим назначение многих из них. Список отсортирован по алфавиту.

The screenshot shows a terminal window titled "sergely : bash — Konsole". The window contains the output of the command "ls -l /etc/". The output lists numerous files and directories in the /etc/ directory, including "aclocal_dirlist", "adjtime", "mkonadi", "aliases", "aliases.d", "aliases.db", "alsa-pulse.conf", "alternatives", "apache2", "arraymog", "arraymog.d", "asound-pulse.conf", "at.deny", "at-spi2", "audisp", "audit", "autofs_ldap_auth.conf", "auto.master", and "auto.master.d". The files are sorted by name.

```
sergely@dhcppc0:~$ ls -l /etc/
итого 2504
-rw-r--r-- 1 root root          25 июл 22 15:03 aclocal_dirlist
-rw-r--r-- 1 root root          18 авг 15 13:21 adjtime
drwxr-xr-x 1 root root          82 авг 15 13:09 mkonadi
-rw-r--r-- 1 root root         2579 окт 13 2015 aliases
drwxr-xr-x 1 root root          0 сен 30 2015 aliases.d
-rw-r--r-- 1 root root        12288 авг 15 13:21 aliases.db
-rw-r--r-- 1 root root         279 ноя 25 2015 alsa-pulse.conf
drwxr-xr-x 1 root root         2216 авг 16 09:04 alternatives
drwxr-xr-x 1 root root          766 авг 16 02:54 apache2
drwxr-xr-x 1 root root          198 авг 16 02:49 arraymog
drwxr-xr-x 1 root root         1520 авг 16 08:34 arraymog.d
-rw-r--r-- 1 root root         263 ноя 25 2015 asound-pulse.conf
-rw-r----- 1 root root          62 окт 19 2015 at.deny
drwxr-xr-x 1 root root          36 окт 29 2015 at-spi2
drwxr-x--- 1 root root          42 окт 29 2015 audisp
drwxr-x--- 1 root root          58 окт 29 2015 audit
-rw----- 1 root root        232 май 31 2016 autofs_ldap_auth.conf
-rw-r--r-- 1 root root        798 май 31 2016 auto.master
drwxr-xr-x 1 root root          0 май 31 2016 auto.master.d
```

/etc/adjtime

Этот конфигурационный файл отвечает за настройку формата системного времени и читается службой `systemd-timedated`.

Время может быть представлено в двух вариантах: LOCAL - время текущего часового пояса и UTC - время по Гринвичу.

Вы можете вручную менять значение или воспользоваться утилитой `timedatectl`.

sergiy : sudo — Konsole

Файл Правка Вид Закладки Настройка Справка

```
0.0 0 0.0
0
LOCAL
```

"/etc/adjtime" 3L, 18C 3,5 Весь

sergiy : sudo

/etc/bash.bashrc

Этот файл принадлежит командной оболочке bash.

Это не совсем конфигурационный файл - а скрипт, его содержимое выполняется при запуске каждого экземпляра bash для настройки оболочки. Точно так же выполняется содержимое файла `~/.bashrc` для каждого пользователя.

Файл Правка Вид Закладки Настройка Справка

```
# /etc/bash.bashrc for SUSE Linux

# PLEASE DO NOT CHANGE /etc/bash.bashrc There are chances that your changes
# will be lost during system upgrades. Instead use /etc/bash.bashrc.local
# for bash or /etc/ksh.kshrc.local for ksh or /etc/zsh.zshrc.local for the
# zsh or /etc/ash.ashrc.local for the plain ash bourne shell for your local
# settings, favourite global aliases, VISUAL and EDITOR variables, etc ...

# Check which shell is reading this file
# ...

if test -z "$is" ; then
    if test -f /proc/mounts ; then
        if ! is=$(readlink /proc/$$/exe 2>/dev/null) ; then
            case "$0" in
                *pcksh)      is=ksh  ;;
                *)          is=sh   ;;
            esac
        fi
        case "$is" in
```

1,1

Наверху



sergiy : sudo

/etc/crontab

Crontab - файл настройки планировщика cron.

Здесь записываются все задания, которые должен выполнить планировщик, а также время и периодичность.

Этот файл не принято редактировать напрямую.

Для этого используется утилита **crontab -e**.

sergely : sudo — Konsole

Файл Правка Вид Закладки Настройка Справка

```
SHELL=/bin/sh
PATH=/usr/bin:/usr/sbin:/sbin:/bin:/usr/lib/news/bin
MAILTO=root
#
# check scripts in cron.hourly, cron.daily, cron.weekly, and cron.monthly
#
*/15 * * * * root test -x /usr/lib/cron/run-crons && /usr/lib/cron/run-crons >/dev
/null 2>&1
```

"./etc/crontab" 8L, 255C

7,1

Весь



sergely : sudo

/etc/environment

Здесь содержатся переменные окружения, которые будут загружены для каждого сеанса терминала, независимо от того запущен он на локальной машине или по ssh.

Файл читается скриптами Bash во время инициализации оболочки.

Файл Правка Вид Закладки Настройка Справка

```
# This file is parsed by pam_env module
#
# Syntax: simple "KEY=VAL" pairs on separate lines
```

"/etc/environment" 5L, 979

1,1

Весь

sergily : sudo

/etc/fstab

Наверное, все уже знают файл /etc/fstab.

Здесь выполняется настройка монтирования файловых систем во время загрузки.

В современных системах он читается systemd и все записи на ходу транслируются в юнит-файлы, с помощью которых уже выполняется монтирование.

Смотрите также: автоматическое монтирование fstab.

sergily : sudo — Konsole

Файл Правка Вид Закладки Настройка Справка

```
UUID=1b31ffdb-c410-4797-8e85-a9891cd06ba4 swap swap defaults 0 0
UUID=89ec0871-2f55-4dd5-aeb2-c0cc4a67997c / btrfs defaults 0 0
UUID=89ec0871-2f55-4dd5-aeb2-c0cc4a67997c /boot/grub2/1386-pc btrfs subvol=@/boot/grub
2/1386-pc 0 0
UUID=89ec0871-2f55-4dd5-aeb2-c0cc4a67997c /boot/grub2/x86_64-efi btrfs subvol=@/boot/g
rub2/x86_64-efi 0 0
UUID=89ec0871-2f55-4dd5-aeb2-c0cc4a67997c /home btrfs subvol=@/home 0 0
UUID=89ec0871-2f55-4dd5-aeb2-c0cc4a67997c /opt btrfs subvol=@/opt 0 0
UUID=89ec0871-2f55-4dd5-aeb2-c0cc4a67997c /srv btrfs subvol=@/srv 0 0
UUID=89ec0871-2f55-4dd5-aeb2-c0cc4a67997c /tmp btrfs subvol=@/tmp 0 0
UUID=89ec0871-2f55-4dd5-aeb2-c0cc4a67997c /usr/local btrfs subvol=@/usr/local 0 0
UUID=89ec0871-2f55-4dd5-aeb2-c0cc4a67997c /var/crash btrfs subvol=@/var/crash 0 0
UUID=89ec0871-2f55-4dd5-aeb2-c0cc4a67997c /var/lib/libvirt/images btrfs subvol=@/var/l
ib/libvirt/images 0 0
UUID=89ec0871-2f55-4dd5-aeb2-c0cc4a67997c /var/lib/mailman btrfs subvol=@/var/lib/mail
man 0 0
UUID=89ec0871-2f55-4dd5-aeb2-c0cc4a67997c /var/lib/mariadb btrfs subvol=@/var/lib/mari
adb 0 0
UUID=89ec0871-2f55-4dd5-aeb2-c0cc4a67997c /var/lib/mysql btrfs subvol=@/var/lib/mysql
0 0
"/etc/fstab" 21L, 1746C
```

1,1

Наверх

sergily : sudo

/etc/group

В этом файле хранятся все группы пользователей, которые есть в системе.

С помощью него вы можете посмотреть список групп, их идентификаторы или добавить новые.

Но добавлять группы с помощью редактирования файла не принято, для этого есть утилита **usermod**

sergely : sudo — Konsole

Файл Правка Вид Закладки Настройка Справка

```
t:x:25:  
audio:x:17:pulse  
avahi:x:486:  
avahi-autoipd:x:486:  
bin:x::daemon  
cdrom:x:20:  
console:x:21:  
daemon:x:2:  
dialout:x:16:  
disk:x:6:  
floppy:x:19:  
ftp:x:49:  
games:x:40:  
kmem:x:9:  
lock:x:54:  
lp:x:7:  
mail:x:12:postfix  
maildrop:x:99:postfix  
man:x:62:  
messagebus:x:497:  
"/etc/group" 68L, 817C
```

1,1

Наверху



sergely : sudo

/etc/hostname

В этом файле содержится имя хоста, файл будет прочитан во время загрузки системы и указанное имя компьютера установится в системе.

Вы будете его видеть в приглашении ввода терминала или в информации о системе.

sergily : sudo — Konsole

Файл Правка Вид Закладки Настройка Справка

linux-qu5o.suse

```
" /etc/hostname" 1L, 16C           1,1      Весь
```

sergily : sudo

/etc/hosts

Файл /etc/hosts позволяет задавать псевдонимы для различных сетевых узлов.

Таким образом, компьютер не обращается к DNS для получения IP домена, а берет его из hosts.

Это позволяет, например, заблокировать доступ к нежелательным сайтам просто перенаправив их на localhost или же получить доступ к сайту по ip, которому еще не присвоен домен

Файл Правка Вид Закладки Настройка Справка

```
# hosts      This file describes a number of hostname-to-address
#           mappings for the TCP/IP subsystem. It is mostly
#           used at boot time, when no name servers are running.
#           On small systems, this file can be used instead of a
#           "named" name server.
#
# Syntax:
#
# IP-Address  Full-Qualified-Hostname  Short-Hostname
#
# special IPv6 addresses
::1          localhost  ipv6-localhost[ipv6-loopback]
fe00::0      ipv6-localnet
ff00::0      ipv6-mcastprefix
"/etc/hosts" 24L, 667C
```

13,1

Наверху



sergly : sudo

/etc/hosts.allow и /etc/hosts.deny

С помощью этих двух файлов можно настраивать права доступа ко всем локальным службам.

Например, вы можете разрешить доступ к службе apache только с локального компьютера.

Это очень сильно повысит безопасность системы, если ваш компьютер подключен к публичной сети.

sergiy : sudo — Konsole

Файл Правка Вид Закладки Настройка Справка

```
# /etc/hosts.deny
# See 'man tcpd' and 'man hosts_access' as well as /etc/hosts.allow
# for a detailed description.

http-gman : ALL EXCEPT LOCAL
```

I

"*/etc/hosts.deny*" 6L, 149C 6,0-1 Весь

sergiy : sudo

/etc/issue и /etc/issue.net

Баннер, который будет выводиться при входе в командную оболочку локально или по SSH.

Обычно там выводится версия ядра и дистрибутива Linux, но вы можете заменить эту информацию по своему усмотрению

sergily : sudo — Konsole

Файл Правка Вид Закладки Настройка Справка

Welcome to openSUSE Leap 42.1 - Kernel \r (\l).

-- ВСТАВКА --

1,1

Весь

sergily : sudo

/etc/ld.so.conf

В этом файле содержатся пути к папкам, в которых компоновщик linux ld.so будет искать динамические библиотеки во время запуска программ.

Папки /lib64, /lib, /usr/lib64 и /usr/lib будут проверены автоматически

/etc/localtime

Это символьическая ссылка, которая указывает на файл часового пояса в папке /usr/share/zoneinfo/.

Редактировать файл не нужно, а для изменения настроек нужно создать символьическую ссылку на другую временную зону

sergiy : bash — Konsole <3>

Файл Правка Вид Закладки Настройка Справка

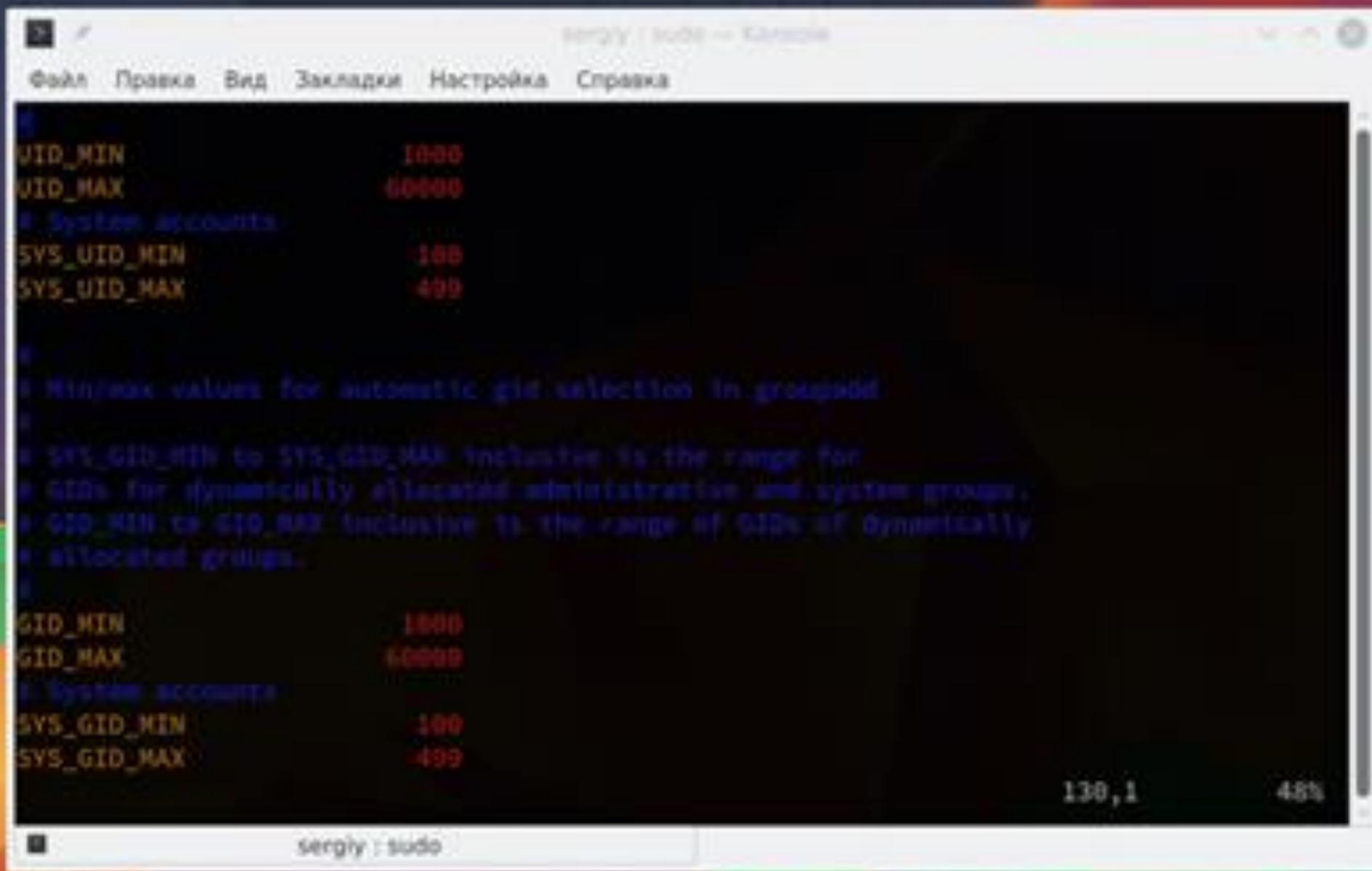
```
sergiy@dhcppc0:~> ls /usr/share/zoneinfo/
Africa      Chile     Factory    Iceland    Mexico     posixrules  US
America     CST6CDT   GB         Indian     MST        PRC          UTC
Antarctica  Cuba      GB-Eire   Iran       MST7MDT   PST8PDT    WET
Arctic       EET       GMT        Israel     Navajo    right       W-SU
Asia         Egypt     GMT0      Israel     NZ         ROC          zone1970.tab
Atlantic     Eire      GMT-8     Jamaica   NZ-CHAT   ROK          zone.tab
Australia   EST       GMT+0     Japan     Pacific   Singapore  Zulu
Brazil       EST5EDT   Greenwich  Kwajalein Poland   Turkey
Canada       Etc       Hongkong  Libya     Portugal  UCT
CET          Europe    HST        MET       posix    Universal
sergiy@dhcppc0:~> █
```

sergiy : bash

/etc/login.defs

Файл `/etc/login.defs` отвечает за настройку поведения утилиты управления пользователями и параметры входа в систему.

Вы можете настроить какой минимальный и максимальный `id` нужно выдавать, что делать с папкой пользователя при удалении и многое другое, количество попыток входа и таймаут, а также многое другое



Energy - audio

/etc/mime.types

В этом файле содержатся общесистемные правила преобразования расширений файлов в понятные системе МИМЕ типы данных.

Затем уже система выбирает, чем открыть тот или иной тип данных.

Файл Правка Вид Закладки Настройка Справка

```
# application/vnd.adobe.partial-upload
application/vnd.adobe.xdp+xml                                xdp
application/vnd.adobe.xfdf                                 xfdf
# application/vnd.aether.imp
# application/vnd.ah-barcode
application/vnd.ahead.space                               ahead
application/vnd.airzip.filesecure.azf                      azf
application/vnd.airzip.filesecure.azs                      azs
application/vnd.amazon.ebook                            azw
application/vnd.americandynamics.acc                     acc
application/vnd.amiga.ami                                ami
# application/vnd.amundsen.maze+xml
application/vnd.android.package-archive                  apk
application/vnd.anser-web-certificate-issue-initiation cii
application/vnd.anser-web-funds-transfer-initiation fti
application/vnd.antix.game-component                   atx
application/vnd.apple.installer+xml                    mpkg
application/vnd.apple.mpegurl                         m3u8
# application/vnd.arastrawi
application/vnd.aristanetworks.swi                      swi
"/etc/mime.types" [только для чтения] 1913L, 102925C          293,1      14%
```

/etc/modprobe.d/

В /etc/modprobe.d/modprobe.conf указываются параметры, с которыми надо грузить модули (файл можно оставить пустым, если железо и так нормально работает)

Чтобы передать параметр модулю ядра, вы можете воспользоваться конфигурационным файлом в modprobe или использовать командную строку ядра с помощью файлов в /etc/modprobe.d/.

Файлы в директории **/etc/modprobe.d/** можно использовать для передачи настроек модуля в udev, который через modprobe управляет загрузкой модулей во время загрузки системы.

Конфигурационные файлы в этой директории могут иметь любое имя, оканчивающееся расширением .conf. Синтаксис следующий:

/etc/modprobe.d/myfilename.conf

options modname parametername=parametervalue

/etc/modprobe.d/

The screenshot shows a terminal window titled "sergily : vi — Konsole". The window contains the following text:

```
#  
# Please don't edit this file. Place your settings into  
# /etc/modprobe.d/99-local.conf instead.  
#  
#  
#####  
# If you want to use the new autofs4  
#####  
  
alias autofs      autofs4  
  
#####  
# block dev aliases  
#####  
  
# network block device  
alias block-major-45      pd  
alias block-major-47      pf
```

The status bar at the bottom right of the terminal window displays "27,0-1" and "7%".

/etc/modules-load.d/

Папка /etc/modules-load.d/ содержит файлы со списками модулей, которые должны быть загружены при запуске системы.

Имя файла не важно, но он должен иметь расширение .conf.

/etc/nsswitch.conf

Этот файл задает настройки порядка разрешения имен в системе для всех программ, написанных на Си или С++.

Например, нужно сначала просматривать локальную сеть и систему, или сразу же отправлять запрос к DNS

passwd: compat
group: compat

hosts: files mdns_minimal [NOTFOUND=return] dns
networks: files dns

services: files
protocols: files
rpc: files
ethers: files
netmasks: files
netgroup: files nis
publickey: files

bootparams: files
automount: files nis
aliases: files

"/etc/nsswitch.conf" 47L, 1223C

46,0-1

Внизу



sergely : sudo

/etc/ntp.conf

Файл ntp.conf отвечает за настройку службы синхронизации времени - ntpd.

В файле указаны адреса ntp серверов, с которых служба будет получать время, а также общие настройки.

Файл Правка Вид Закладки Настройка Справка

```
# Clients from this (example!) subnet have unlimited access, but only if
# it is cryptographically authenticated.
#restrict 192.168.123.0 mask 255.255.255.0 nomask

#*
#* Miscellaneous stuff
#*

driftfile /var/lib/ntp/drift/ntp.drift
# path for drift file

logfile /var/log/ntp
# alternate log file
# logconfig noyncstatus + sysevents
# logconfig nati

# statdir /tmp/          # directory for statistics files
# filegen peerstats file peerstats type file enable
# filegen loopstats file loopstats type file enable
```

1,0-1

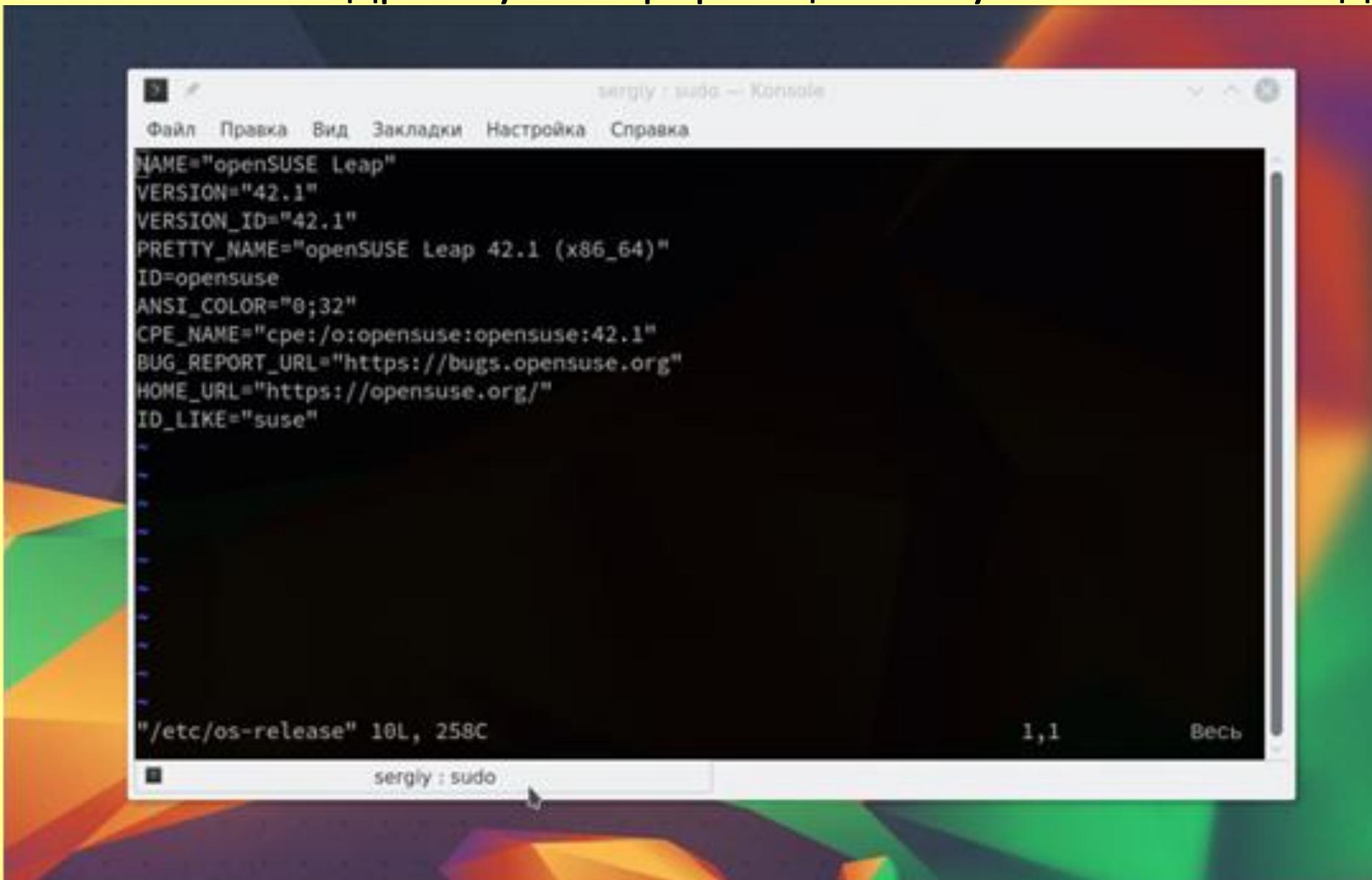
Назерху



sergely : sudo

etc/os-release

Отображает очень подробную информацию об установленном дистрибутиве



The screenshot shows a terminal window titled "загрузка - мифо - Консоль" (загрузка - myfo - Console) running on a desktop environment with a colorful abstract background. The terminal displays the following text:

```
NAME="openSUSE Leap"
VERSION="42.1"
VERSION_ID="42.1"
PRETTY_NAME="openSUSE Leap 42.1 (x86_64)"
ID=opensuse
ANSI_COLOR="0;32"
CPE_NAME="cpe:/o:opensuse:opensuse:42.1"
BUG_REPORT_URL="https://bugs.opensuse.org"
HOME_URL="https://opensuse.org/"
ID_LIKE="suse"

"/etc/os-release" 10L, 258C           1,1      Весь
```

The terminal prompt at the bottom is "sergely : sudo".

/etc/passwd

Файл содержит список всех зарегистрированных в системе пользователей, а также дополнительные настройки для них, например, оболочку, дату смены пароля и дату отключения аккаунта, кроме самого пароля.

Напрямую файл лучше не редактировать, а использовать утилиту для управления пользователями `adduser` или `deluser`.

/etc/profile

Файл `/etc/profile`, точно так же как и `/etc/environment` загружается и выполняется при запуске любой командной оболочки в системе.

Но в отличие от `environment`, это скрипт, а значит, он может задавать не только переменные, но и выполнять различные команды для инициализации оболочки

Файл Правка Вид Закладки Настройка Справка

/etc/profile for Slack Linux

```
# PLEASE DO NOT CHANGE /etc/profile. There are chances that your changes  
# will be lost during system upgrade. Instead use /etc/profile.local for  
# your local settings, favour the global aliases, PATH and export  
# variables, etc ...
```

```
#  
# Check what shell is reading this file
```

```
if test -f /proc/mounts ; then  
    if [ $(readlink /proc/$$/exe) != /dev/null ] ; then  
        case "$0" in  
            *pcksh)      ts=ksh   ;;  
            *)          ts=sh   ;;  
        esac  
    fi  
    case "$ts" in  
        */bash)     ts=bash  
        case "$0" in  
            "/etc/profile" 386L, 9168C
```

1,1

Наверху



sergby : sudo

/etc/resolv.conf

В этом файле содержатся IP адреса DNS серверов, которые будет использовать компьютер.

В большинстве дистрибутивов вы можете редактировать файл вручную или же использовать специальные утилиты

sergily : sudo — Konsole

Файл Правка Вид Закладки Настройка Справка

```
# Generated by NetworkManager
nameserver 192.168.1.1
```

I

"/etc/resolv.conf" 2L, 53C 1,18 Весь

sergily : sudo

/etc/sddm.conf

Это конфигурационный файл Linux для настройки менеджера входа sddm, для других менеджеров входа будут свои файлы настройки.

Здесь можно изменить максимальный и минимальный ID пользователя, который может войти в систему, например, чтобы разрешить авторизацию root, изменить тему, добавить вход без пароля и многое другое.

Файл Правка Вид Закладки Настройка Справка

[Theme]

Current=breeze

CursorTheme=breeze_cursors

[XDisplay]

ServerPath=/usr/bin/X

SessionCommand=/etc/X11/xdm/Xsession

DisplayCommand=/etc/X11/xdm/Xsetup

MinimumVT=7

[Autologin]

Autologin user

User=sergiy

I

"/etc/sddm.conf" 13L, 210C

1,1

Весь



sergely : sudo

/etc/shadow

Раньше пароли пользователя содержались в файле /etc/passwd, но поскольку к нему мог получить доступ любой пользователь, это было небезопасно, несмотря на то, что пароли зашифрованы.

Поэтому все пароли были вынесены в /etc/shadow. Вы можете изменить пароль пользователя.

Файл Проверка Вид Закладки Настройка Справка

```
bt:::17028:::::  
avahi:::17038:::::  
avahi-autoipd:::17028:::::  
bin:::16737:::::  
daemon:::16737:::::  
dnsmasq:::17028:::::  
ftp:::16737:::::  
ftpsecure:::17028:::::  
games:::16737:::::  
lp:::16737:::::  
mail:::16737:::::  
man:::16737:::::  
messagebus:::16737:::::  
mysql:::17028:::::  
news:::16737:::::  
nm-openvpn:::17028:::::  
nobody:::16737:::::  
nscd:::16737:::::  
ntp:::16737:::::  
polkitd:::16737:::::
```

1,1

Наверху



sergely : sudo

/etc/sudoers

/etc/sudoers - это файл настройки прав доступа к утилите sudo.

Эта утилита позволяет выполнять команды от имени других пользователей, в том числе от имени суперпользователя.

Но использовать ее могут только те пользователи, которые прописаны в этом файле.

Файл Правка Вид Закладки Настройка Справка

```
cat
# This comment to enable logging of a command's output, except for
# xaudoreplay and reboot. Use audoreplay to play back logged sessions.
Defaults log_output
Defaults:/usr/bin/audoreplay log_output
Defaults:/sbin/reboot log_output

# In the default (unconfigured) configuration, sudo asks for the root password.
# This allows use of an ordinary user account for administration of a freshly
# installed system. When configuring sudo, delete the two
# following lines:
Defaults targetpw # ask for the password of the target user i.e., root
ALL      ALL=(ALL) ALL # WARNING! Only use this together with 'Defaults targetpw'

#  
# Runas alias specification  
#  
#
```

78,1

79%



sergely : sudo

/etc/sysctl.conf

Этот файл отвечает за настройку параметров ядра во время выполнения.

Тут вы можете задать все параметры из подсистемы /sys/ и они будут со-хранены после перезагрузки

Файл Правка Вид Закладки Настройка Справка

```
# sysctl reads settings from the following locations:
# /boot/sysctl.conf<kernelversion>
# /lib/sysctl.d/*.conf
# /usr/lib/sysctl.d/*.conf
# /usr/local/lib/sysctl.d/*.conf
# /etc/sysctl.d/*.conf
# /run/sysctl.d/*.conf
# /etc/sysctl.conf

# To disable or override a distribution provided file just place a
# file with the same name in /etc/sysctl.d/
#
# See sysctl.conf(5), sysctl.d(5) and sysctl(8) for more information
#
# See https://www.freedesktop.org/wiki/Software/systemd/sysctl.d/ for
# details on how to use the .d/ mechanism.

kernel.sysrq = 0
net.ipv4.ip_forward = 0
net.ipv4.tcp_syncookies = 1
net.ipv6.conf.all.forwarding = 0
```

23,1

Внизу



sergely : sudo

/etc/vconsole.conf

У этого файла только одна цель - задать кодировку, раскладку клавиатуры и шрифт по умолчанию для всех виртуальных консолей, запускаемых на машине

sergely : sudo — Konsole

Файл Правка Вид Закладки Настройка Справка

```
KEYMAP=ruwin_alt-UTF-8
```

" /etc/vconsole.conf " 1L, 23C 1,1 Весь

sergely : sudo

/boot/grub/grub.cfg

Этот конфигурационный файл Linux находится не в /etc из-за своего особого предназначения.

Здесь содержатся все настройки загрузчика, пункты меню и другие параметры, поэтому он должен быть доступен еще до того как была подключена корневая файловая система.

sergily : sudo — Konsole

Файл Правка Вид Закладки Настройка Справка

```
## BEGIN /etc/grub.d/10_Linux ##  
menuentry 'openSUSE Leap 42.1' --class opensuse --class gnu-linux --class gnu --class  
os $menuentry_id_option 'gnulinux-simple-09ec0871-2f55-4dd5-aeb2-cacc4a67907c' {  
    load_video  
    set gfxpayload=keep  
    insmod gzio  
    insmod part_gpt  
    insmod btrfs  
    set root='hd0,gpt3'  
    if [ $feature_platform_search_hint = xy ]; then  
        search --no-floppy --fs-uuid --set=root --hint-bios=hd1,gpt3 --hint-efi=hd1,  
        gpt3 --hint-baremetal=ahci1,gpt3 --hint='hd0,gpt3' 09ec0871-2f55-4dd5-aeb2-cacc4a6790  
        7c  
    else  
        search --no-floppy --fs-uuid --set=root 09ec0871-2f55-4dd5-aeb2-cacc4a67907c  
    fi  
    echo 'Загружается Linux 4.1.27-27-default ...'  
    linux /boot/vmlinuz-4.1.27-27-default root=UUID=09ec0871-2f55-4dd5-aeb2-cacc  
    4a67907c ${extra_cmdline} resume=/dev/sdb2 splash=silent quiet showopts
```

144,1-8

57%

sergily : sudo

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`”.

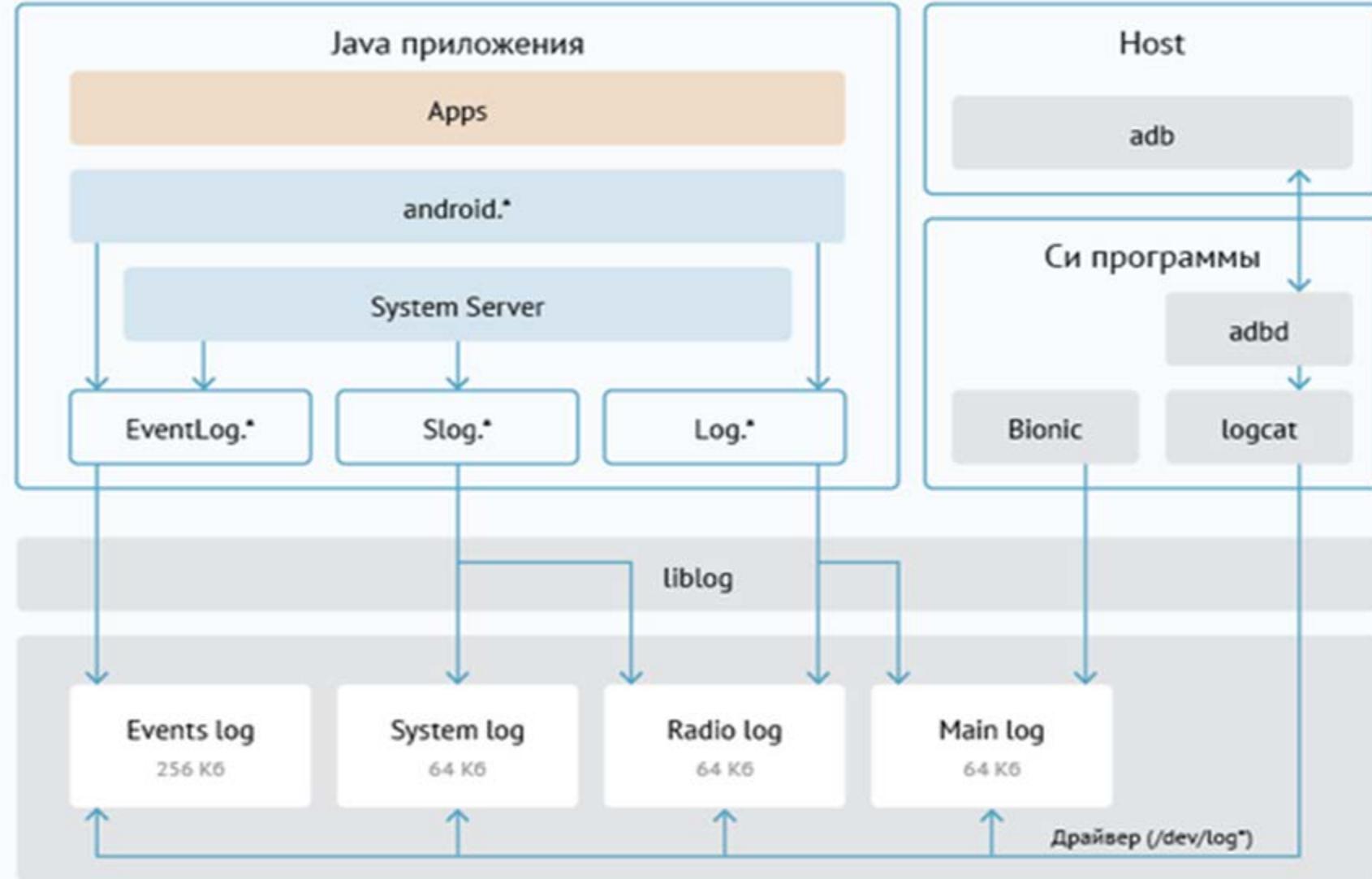
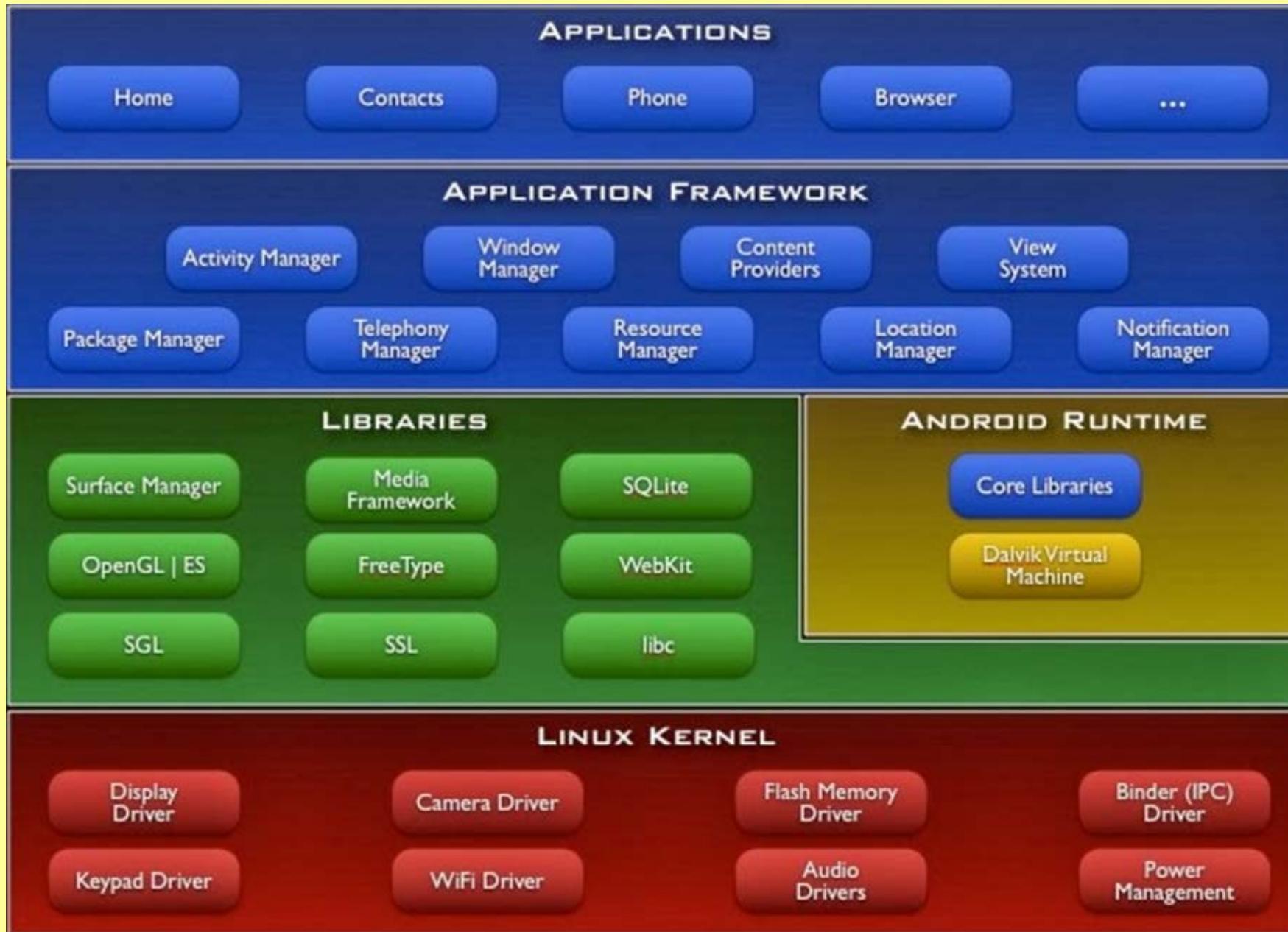
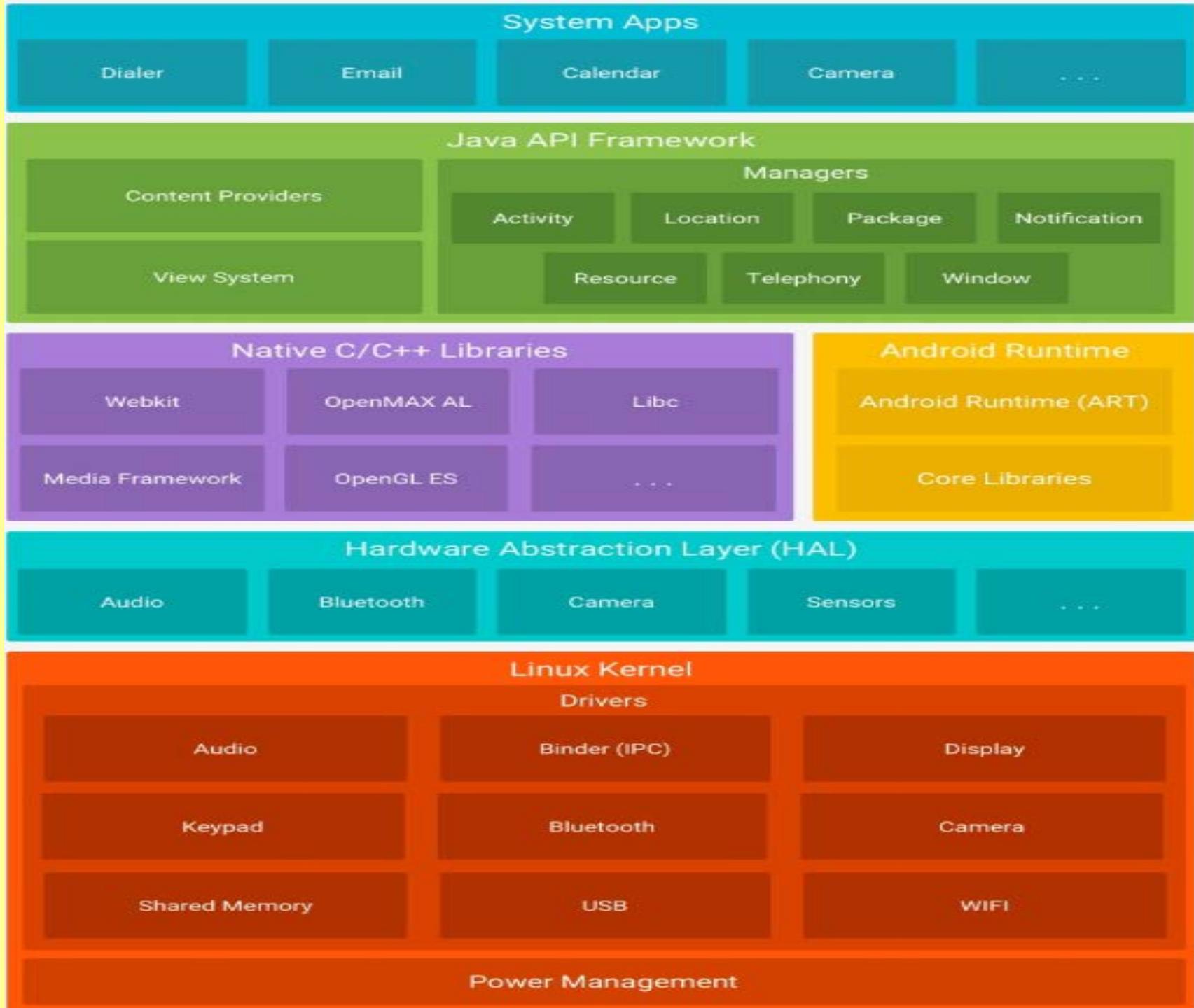


Диаграмма - общая система логирования Андроида

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.

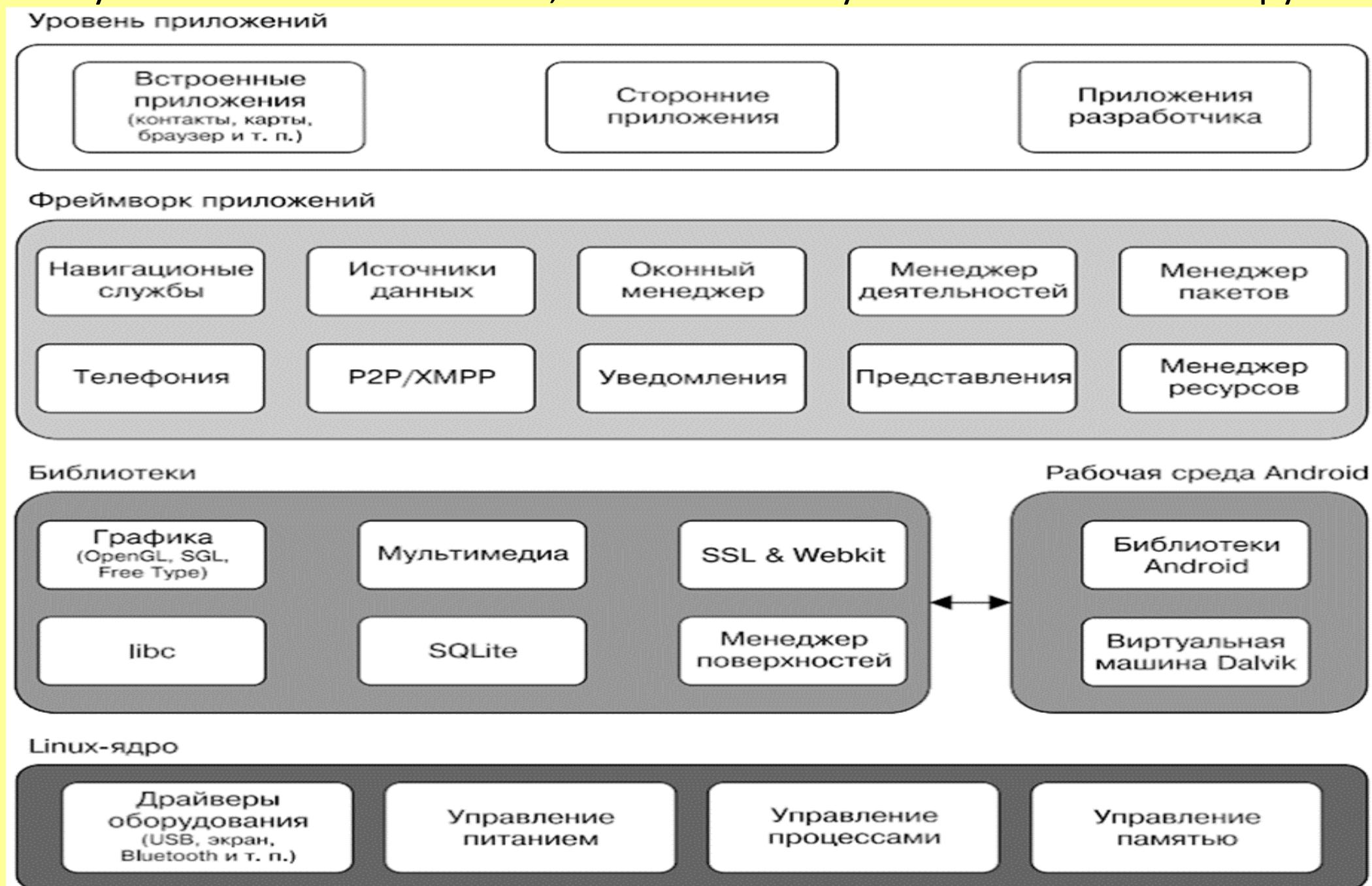
Короче говоря, это делается для того, чтобы избежать лицензии GPL на ядро Linux.

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 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(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.

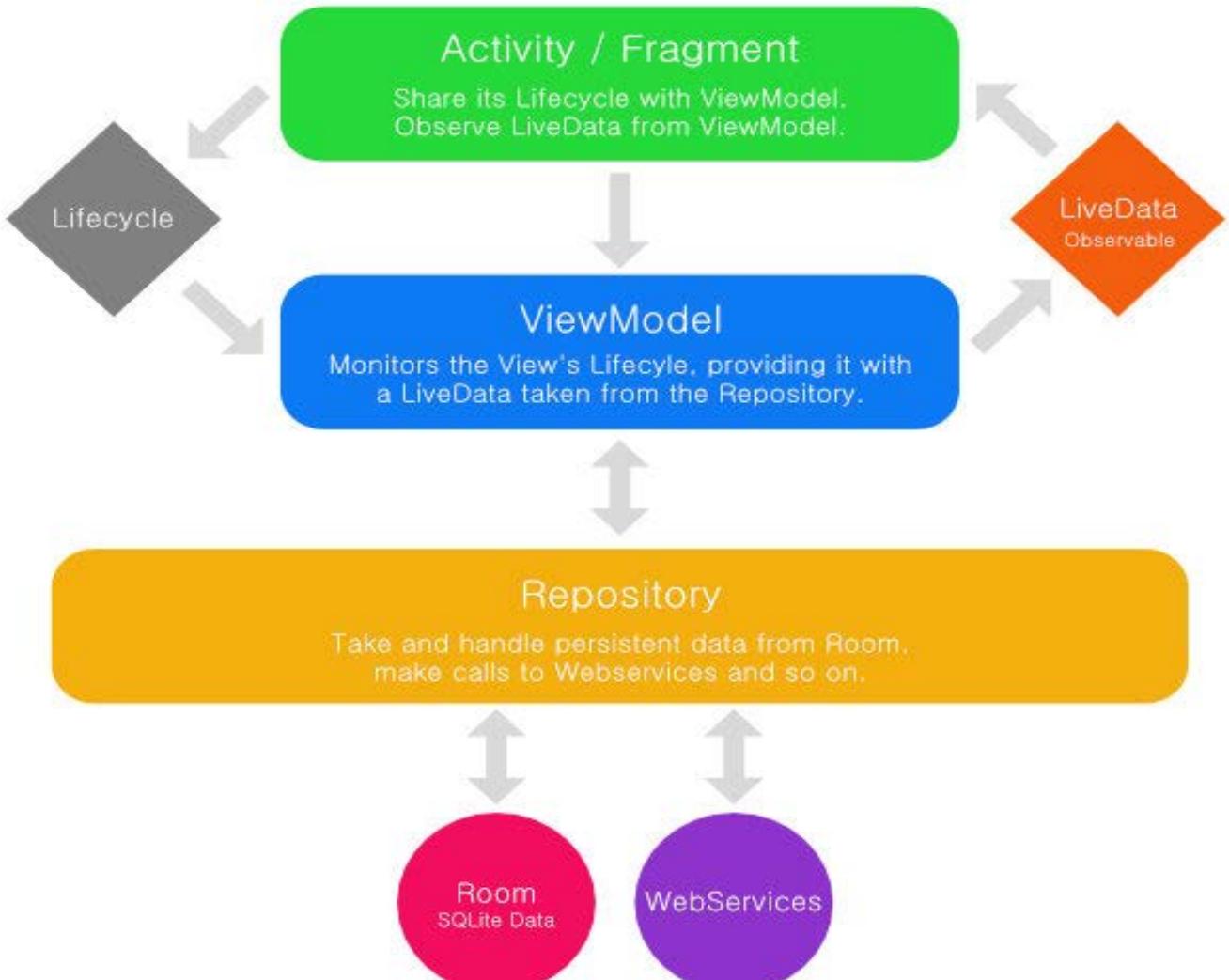
Это, однако, не имеет значения, так как важно то, что он полагается на новые компоненты архитектуры, чтобы создать разделение задач с отличной тестируемостью и ремонтопригодностью.

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>

Лекция

Веб-сервер

Вопросы рассматриваемые на лекции

HTTP-запросы и ответы. HTTP-протокол.

Структура HTTP-запроса. Структура запроса. CRLF.

Заголовки запроса. Request-заголовки.

Протокол HTTPS. Ответы сервера.

Веб-сервер. Ресурсы. Сервлет. Установка Tomcat в системе Windows. Установка Tomcat в ОС Linux (Ubuntu).

Фреймворк Spring и Eclipse. Создание в Eclipse Spring-проекта.
Обработка запросов на сервере.

Реализация сервера на PHP. Как установить Apache и PHP?

HTTP-протокол

Сегодня любой потребитель интернет-ресурсов сталкивается с HTTP-протоколом. Этот сетевой протокол, будучи разработанным в 90-х годах, и до сегодняшнего дня является основным для передачи информации в мировой сети.

HTTP (HyperText Transfer Protocol) — протокол передачи гипертекста, протокол прикладного уровня по классификации ISO-OSI.

Сегодня чаще всего используется версия 1.1 протокола.

Полное описание протокола можно увидеть в документе RFC 2616.

При взаимодействии в интернете участников взаимодействия обычно разделяют на две роли — клиент и сервер.

Где сервер (serv — обслуживание) — поставщик информации/услуг, а клиент — потребитель. HTTP также предполагает клиент-серверное взаимодействие.

То есть программа-клиент (например, браузер) отправляет HTTP-запрос, сервер его принимает, обрабатывает и отправляет клиенту HTTP-ответ.

Таким образом, взаимодействие клиента и сервера по HTTP похоже на обмен почтовыми письмами через службу доставки.

Несмотря на то что в WWW протокол используется для передачи текстовой и медиаинформации, многие API используют его для передачи данных.

При этом данные чаще всего кодируются как строковые, или в более универсальном представлении как JSON/XML.

Структура HTTP-запроса

Рассмотрим, каким образом происходит обмен между клиентом и сервером по HTTP.

При этом каждый запрос порождает следующую последовательность шагов.

1. DNS-запрос — поиск ближайшего DNS-сервера, чтобы из URL (например, yandex.ru) получить реальный IP-адрес.
2. Установка TCP-соединения с сервером по полученному IP-адресу.
3. Отправка данных HTTP-запроса.
4. Ожидание ответа, пока пакеты дойдут до сервера, он их обработает и вернет ответ назад.
5. Получение данных HTTP-ответа. Первые две операции, как правило, занимают больше всего времени.

Структура запроса

Каждый HTTP-запрос состоит из трех частей, которые следуют в указанном порядке.

1. Стока запроса — указан метод запроса (HTTP-метод), URI, версия протокола.
2. Заголовки — характеризуют тело сообщения, параметры передачи и прочие сведения.
3. Тело сообщения — данные сообщения.

HTTP-спецификация определяет более строгое описание структуры запроса или ответа:

message = <start-line>

*(<message-header>)

CRLF

[<message-body>]

<start-line> = Request-Line | Status-Line

<message-header> = Field-Name `:` Field-Value ``

CRLF — обязательная «новая» строка между секцией заголовка и телом.

Заголовок (message-header) и тело (message-body) запроса может отсутствовать, но строка запроса (start-line) есть всегда.

Строка запроса. Методы GET и POST

Строка запроса состоит из трех разделов, разделенных пробелом — Method URI Protocol.

Рассмотрим пример запроса, который может использовать браузер при обращении к странице `http://myitschool.ru/book` — GET /book HTTP/1.1.

В этом примере через пробел указываются метод GET, URI, указывающий нужный раздел сайта и версию протокола:

Метод	URI	Протокол
GET	/book	HTTP/1.1

В поле метод указывается операция, которую нужно осуществить с указанным ресурсом.

Сервер может выполнять тот набор методов, который в нем запрограммирован. Однако в большинстве фреймворков реализованы следующие методы:

- GET — получить ресурс. При этом URI содержит информацию, позволяющую найти ресурс;
- POST — создать новый ресурс;
- PUT — обновить существующий ресурс;
- DELETE — удалить существующий ресурс.

Рассмотрим два наиболее распространенных метода более подробно.

- Метод GET предназначен для получения требуемой информации и передачи данных в адресной строке.

Удобство использования метода GET заключается в том, что адрес со всеми параметрами можно использовать неоднократно, сохранив его, например, в закладки браузера, а также менять значения параметров прямо в адресной строке.

- Метод POST посылает на сервер данные в запросе браузера. Это позволяет отправлять большее количество данных, чем доступно методу GET, поскольку у него установлено ограничение в 4 Кб.

Большие объемы данных используются в форумах, почтовых службах, заполнении базы данных, при пересылке файлов и др.

Таким образом между методами POST и GET следующие различия (см. табл. 5.1).

Характеристика	GET	POST
Тело запроса	Отсутствует	Содержит данные
Максимальный объем	255 символов	8 КБ
Кэширование	Да	Нет

Табл. 5.1.

Кроме того, следует помнить, что передача дополнительных параметров у GET и POST запросов происходит по-разному.

У POST-запроса дополнительные параметры передаются в теле запроса, а у GET-запроса дополнительные параметры передаются в URL.

Синтаксис следующий:

URI[?param1=value1[¶mN=valueN]]

Например, если в браузере набрать адрес
<https://www.google.ru/search?q=myitschool+rostov&ie=utf-8>, то браузер
отправит серверу google.ru следующий GET-запрос:

GET /search?q=myitschool+rostov&ie=utf-8 HTTP/1.1

Параметры этого запроса следующие:

Имя параметра	Значение
q	myitschool+rostov
ie	utf-8

Не следует передавать в параметрах GET-запросов конфиденциальных
данных, так как они присутствуют в запросе в открытом виде и будут
легкодоступны посторонним. Например, в истории просмотра браузера или в
каких-либо других журналах.

Заголовки запроса

Заголовки запроса задают его параметры. В зависимости от назначения заголовки HTTP-запросов разделяют на 4 группы:

- general;
- request specific;
- response specific;
- entity.

Из **General-заголовков** (применяются как для запроса, так и ответа сервера) наиболее распространенные:

- Date указывает дату запроса, пример: Date: Fri, 29 Jun 2016 09:20:45 GMT;
- MIME-version указывает версию MIME (по умолчанию 1.0), пример: MIME-version: 1.0;
- Pragma содержит указания для таких промежуточных агентов, как прокси и шлюзы, пример: Pragma: no-cache.

Request-заголовки:

- Host содержит доменное имя вебсервера, например: Host: www.google.ru;
- Authorization содержит информацию об аутентификации, например: Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==;
- From — поле, в котором браузер может посыпать полный e-mail адрес пользователя серверу, например: From: info@myitschool.ru;
- User-Agent указывает наименование и версию браузера, ОС, например: User-Agent: Mozilla/3.0.

Протокол HTTPS

Сам по себе протокол HTTP не предполагает использование шифрования для передачи информации.

Однако для HTTP есть распространенное расширение, которое реализует упаковку передаваемых данных в криптографический протокол SSL или TLS. Название этого расширения — HTTPS (HyperText Transfer Protocol Secure).

В отличие от стандартного для HTTP 80-го порта, для HTTPS обычно используется TCP-порт 443.

HTTPS широко используется для защиты информации от перехвата, а также, как правило, обеспечивает защиту от атак вида man-in-the-middle — в том случае, если сертификат проверяется на клиенте, и на компьютере пользователя не были внедрены сертификаты центра сертификации злоумышленника.

На данный момент HTTPS поддерживается всеми популярными веб-браузерами (Протокол HTTP/HTTPS

Пример

Хороший способ познакомиться с протоколом HTTP — это поработать с каким-нибудь веб-ресурсом вручную.

Для этого необходимо воспользоваться любой подходящей утилитой командной строки, например, telnet.

В ОС Windows, возможно, придется ее установить. Как это сделать описано здесь: <http://windows.microsoft.com/ru-ru/windows/telnet-faq>.

В качестве примера сымитируем запрос браузером страницы - <http://myitschool.ru/cert.php>. Для этого откроем окно командной оболочки (в Windows WIN+r -> cmd -> <Enter>). В открывшемся окне оболочки запустим команду telnet myitschool.ru 80. В открывшемся соединении к серверу введем следующие строки HTTP протокола:

- GET /cert.php HTTP/1.1 < Enter>
- host: myitschool.ru <Enter>
- <Enter>

В случае успешного ответа сервера (код 200) в консоль будет выведен большой HTML-документ. Для закрытия соединения введите CTRL+] и далее exit (см. рис. 5.8).

```
Terminal - root@node:~  
File Edit View Terminal Tabs Help  
node:~ # telnet myitschool.ru 80  
Trying 194.135.112.158...  
Connected to myitschool.ru.  
Escape character is '^]'.  
GET /cert.php HTTP/1.1  
host: myitschool.ru  
  
HTTP/1.1 200 OK  
Server: nginx/1.8.0  
Date: Sun, 04 Mar 2018 12:54:42 GMT  
Content-Type: text/html; charset=UTF-8  
Transfer-Encoding: chunked  
Connection: keep-alive  
Keep-Alive: timeout=20  
Vary: Accept-Encoding  
  
1f57  
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">  
  
<meta charset="UTF-8" xmlns="http://www.w3.org/1999/html"/>  
  
<html class="ltr" xmlns="http://www.w3.org/1999/xhtml">  
<head>  
    <title>  
        IT школа Samsung  
    </title>
```

Рис. 5.8

В приведенном примере производился HTTP обмен через сетевое соединение, которое было открыто к серверу при помощи программы telnet.

Можно воспользоваться командой netstat, чтобы увидеть список текущих соединений.

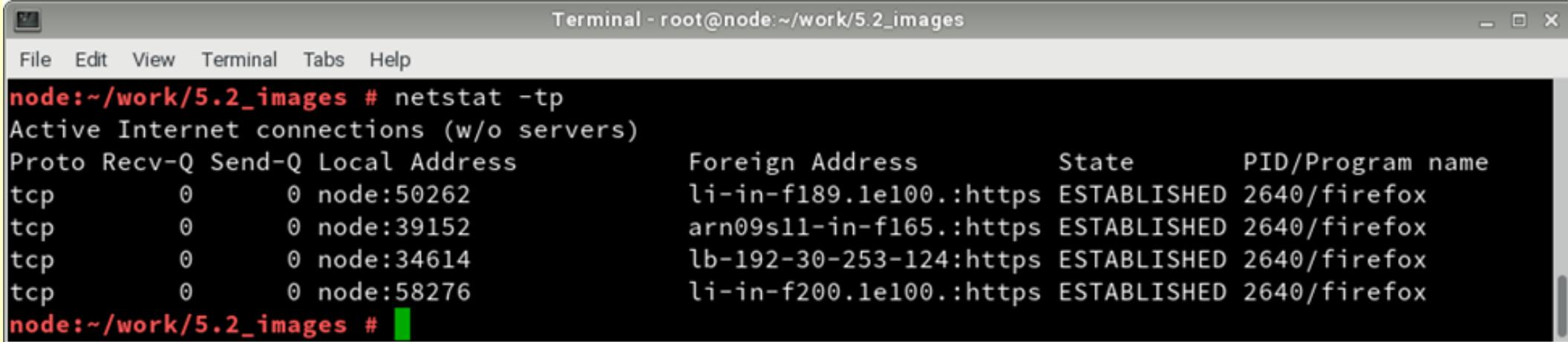
Для этого необходимо в системе Windows набрать:

```
netstat -f -o
```

Для ОС Linux, Mac и т. д. команда следующая:

`netstat -tp`

Будет выведен список TCP-соединений, причем, в столбце Local Address будет указан IP-адрес вашего компьютера и через двоеточие порт, выделенный ОС, с которым связан идентификатор работающего программного процесса (последний столбец) (см. рис. 5.9).



The screenshot shows a terminal window titled "Terminal - root@node:~/work/5.2_images". The window has a standard Linux-style title bar with icons for minimize, maximize, and close. Below the title bar is a menu bar with "File", "Edit", "View", "Terminal", "Tabs", and "Help". The main area of the terminal displays the output of the "netstat -tp" command. The output is as follows:

```
node:~/work/5.2_images # netstat -tp
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address          Foreign Address        State      PID/Program name
tcp        0      0  node:50262              li-in-f189.1e100.:https ESTABLISHED 2640/firefox
tcp        0      0  node:39152              arn09s11-in-f165.:https ESTABLISHED 2640/firefox
tcp        0      0  node:34614              lb-192-30-253-124:https ESTABLISHED 2640/firefox
tcp        0      0  node:58276              li-in-f200.1e100.:https ESTABLISHED 2640/firefox
node:~/work/5.2_images #
```

Рис. 5.9.

Эта пара IP-адрес и порт и задают клиентский сокет (socket — розетка, разъем). Когда с сервера на указанный порт приходит ответ, операционная система ищет связанный с ним сокет и помещает в него данные.

На сервере сокет работает по другому механизму. Приложение имеет возможность создать «слушающий» сокет и, привязав его к порту ОС сервера, получать адресованные ему пакеты данных.

Ответы сервера

Познакомимся с тем, как внутри устроены запросы и ответы к серверу. Для этого откроем страницу ya.ru (или любую другую на выбор). Далее нужно открыть в настройках браузера «Developer Tools» (Ctrl+Shift+I в Chrome, F12 в IE), перейти в вкладку «Network» (в IE еще необходимо нажать кнопку «Start Capture» на панели). Обновим текущую страницу, чтобы появилась информация. Будет выведена табличка, в которой каждая строчка — это некоторый запрос к серверу. В современном мире даже самая простая страница отправляет десяток запросов к серверу (рис. 5.10).

The screenshot shows the Microsoft Internet Explorer Developer Tools interface. The top menu bar includes File, Find, Disable, View, Images, Cache, Tools, Validate, and tabs for Browser Mode: IE9 and Document Mode: IE9 standards. Below the menu is a toolbar with icons for HTML, CSS, Console, Script, Profiler, and Network, with the Network tab currently selected. A sub-toolbar below the main toolbar includes icons for Stop capturing and Go to detailed view. The main area displays a table of network requests:

URL	Method	Result	Type	Received	Taken	Initiator	Timings
https://ya.ru/	GET	200	text/html	12,96 KB	250 ms	refresh	
https://yastatic.net/www/_/t/f/LT2KjI4uj...	GET	304	text/css	302 B	< 1 ms	<link rel="style..."	
https://yastatic.net/www/_/j/k/kNvXxWXf...	GET	304	text/css	301 B	< 1 ms	<link rel="style..."	
https://yastatic.net/jquery/1.8.3/jquery....	GET	304	application/x-java...	311 B	< 1 ms	<script>	
https://yastatic.net/www/2.666/white/pa...	GET	304	application/x-java...	319 B	< 1 ms	<script>	
https://mc.yandex.ru/metrika/watch.js	GET	304	application/x-java...	254 B	< 1 ms	<script>	
https://yastatic.net/www/_/c/Q/WXXr7gj...	GET	304	image/svg+xml	307 B	< 1 ms	background-image	
https://yastatic.net/islands/_/K4e4uv8u9I...	GET	304	image/svg+xml	280 B	< 1 ms	background-image	

Рис. 5.10

В этом примере видно, как и какие HTTP-запросы (request) отправляются к веб-серверу.

И здесь же видно полученные от сервера HTTP-ответы (response). HTTP-ответ устроен примерно, как и запрос.

В теле ответа можно увидеть получаемую от сервера веб-страницу, а в заголовках ответа есть очень важное поле — код статуса ответа (status).

Этот код помогает клиенту понять, как именно интерпретировать ответ сервера, а также предусмотреть обработку нестандартных ситуаций.

Спецификация HTTP определяет трехзначные коды, описывающие состояния HTTP-ответа, которое получено от сервера.

Причем они объединены в 5 групп, начинающихся с 1 до 5. В таблице 5.2 ниже приведены наиболее часто используемые коды.

Код	Описание кода состояния HTTP
1xx	Информационные коды. Этот класс состояний был добавлен в HTTP/1.1 и носит условный характер. Например, сервер может послать заголовок Expect: 100-continue, что будет означать для клиента сигнал к продолжению отправки оставшейся части запроса. Если запрос отправлен полностью, то клиент проигнорирует этот заголовок. HTTP/1.0 -клиенты проигнорирует его в любом случае
2xx	Успешное выполнение запроса. Эта группа состояний говорит клиенту, что все прошло хорошо и запрос успешно обработан. Чаще всего встречается код 200 OK. В случае GET-метода вместе с таким кодом в теле HTTP-ответа отправляется запрашиваемый ресурс
200	Запрос был обработан успешно
201	Объект создан
202	Информация принята
203	Информация, которая не заслуживает доверия
204	Нет содержимого
205	Сбросить содержимое
206	Частичное содержимое (например, при «докачке» файлов)

	Перенаправление (чтобы выполнить запрос, нужны какие-либо действия). Коды этого диапазона означают, что клиенту необходимо сделать другой запрос, чтобы получить требуемые ресурс. Наиболее частый сценарий — запрос на другой URL
3xx	
300	Несколько вариантов на выбор
301	Ресурс перемещен на постоянной основе
302	Ресурс перемещен временно
303	Смотрите другой ресурс
304	Содержимое не изменилось
305	Используйте прокси-сервер

4xx	Проблема связана не с сервером, а с запросом. Эти коды используются, когда сервер полагает, что в результате запроса клиент допустил ошибку, например, запрашивает несуществующий ресурс. Наиболее распространенный код — 404. С ним сталкивались, пожалуй, все. Он говорит клиенту, что запрошенный ресурс не существует на сервере
400	Некорректный запрос
401	Нет разрешения на просмотр ресурса
402	Требуется оплата
403	Доступ запрещен
404	Ресурс не найден
405	Недопустимый метод
406	Неприемлемый запрос
407	Необходима регистрация на прокси-сервере
408	Время обработки запроса истекло
409	Конфликт
410	Ресурса больше нет
411	Необходимо указать длину
412	Не выполнено предварительное условие
413	Запрашиваемый элемент слишком велик
414	Идентификатор ресурса (URI) слишком длинный
415	Неподдерживаемый тип ресурса

5xx	Ошибки на сервере. Этот класс состояний говорит о серверной ошибке в процессе обработки клиентского запроса. Чаще всего встречается ошибка 50
500	Внутренняя ошибка сервера
501	Функция не реализована
502	Дефект шлюза
503	Служба недоступна
504	Время прохождения через шлюз истекло
505	Неподдерживаемая версия HTTP

Табл. 5.2.

Веб-сервер

Веб-сервер — сервер, принимающий HTTP-запросы от клиентов, обычно веб-браузеров, и выдающий им HTTP-ответы, обычно вместе с HTML-страницей, изображением, файлом или другими данными.

Веб-сервером называют как программное обеспечение, выполняющее функции веб-сервера, так и непосредственно компьютер, на котором это программное обеспечение работает. Клиент передает веб-серверу запросы на получение ресурсов, идентифицируемые по URL-адресами.

Ресурсы — это хранимые на сервере HTML-файлы, изображения, медиапотоки или другие данные, которые необходимы клиенту. В ответ веб-сервер передает клиенту запрошенные данные. Этот обмен происходит по протоколу HTTP.

Существует множество веб-серверов, написанных на разных языках.

Самыми распространенными являются Apache и nginx.

Современные веб-сервера могут выдавать клиенту не только статично хранимые файлы, но и динамически формируемые данные, которые образуются в результате вызова программного кода, написанного на различных языках программирования.

Такой код для языка Java называется сервлетом.

Наиболее распространенный сервер с открытым исходным кодом и поддержкой сервлетов — Tomcat, разрабатываемый Apache Software Foundation.

Чтобы подчеркнуть именно особенность этого сервера, выполняется Java-код при обращении клиента, говорят — Tomcat — контейнер сервлетов, реализующий спецификацию сервлетов и спецификацию JavaServer Pages (JSP) и JavaServer Faces (JSF). Сам Tomcat тоже написан на языке Java.

Установка Tomcat в системе Windows

Чтобы установить Tomcat в ОС Windows, нужно зайти на сайт <http://tomcat.apache.org/> из раздела Downloads выбрать последнюю актуальную версию (на данный момент это Tomcat 9.0), нажать на ссылку 32-bit/64-bit Windows Service Installer в разделе Binary Distributions.

Запускаем скаченный установщик.

На приветственной странице, следуя инструкциям, нажимаем на кнопку Next, в следующем окне принимаем лицензионное соглашение I Agree. В третьем окне выбираем элементы (см. рис. 5.11).

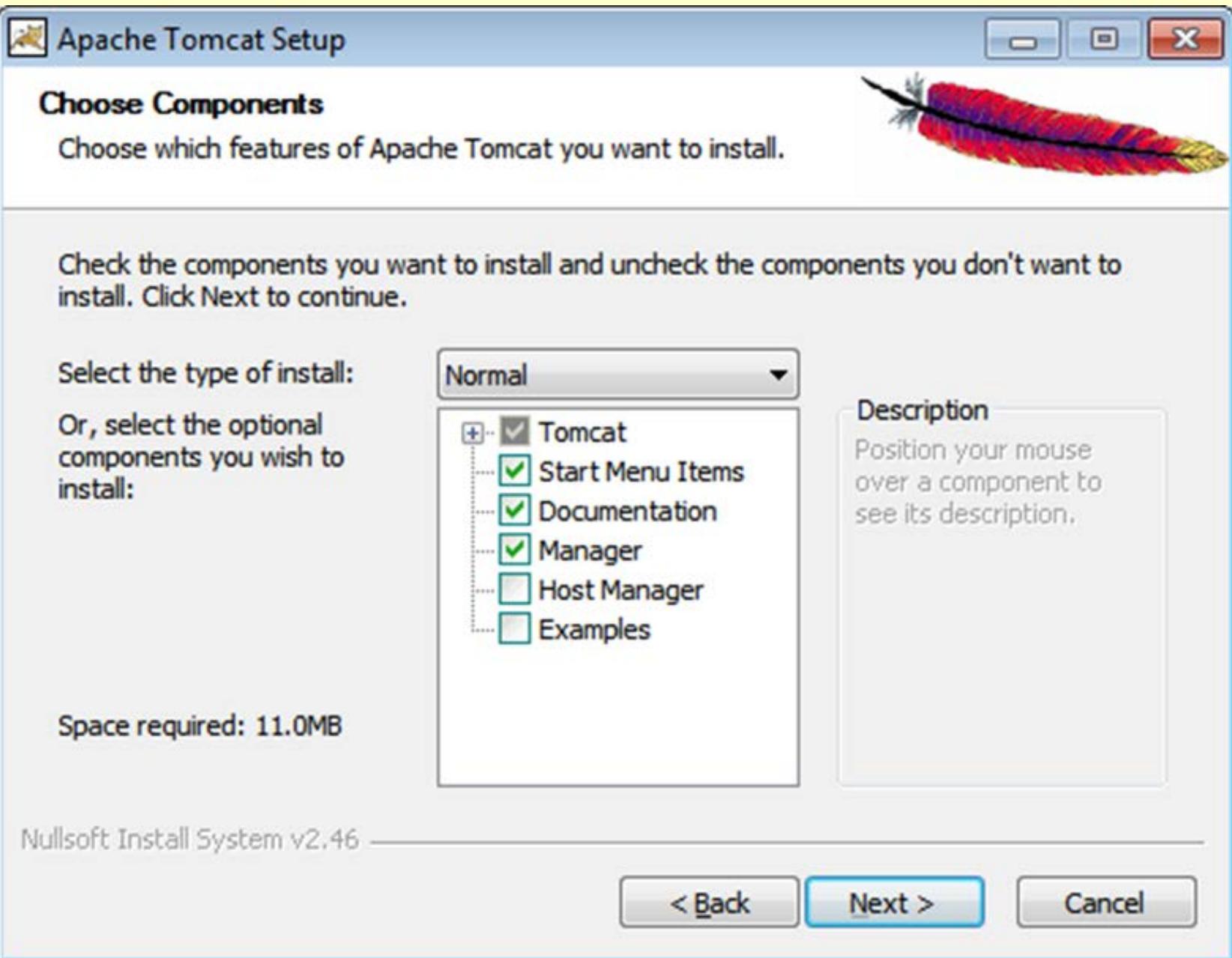


Рис. 5.11

В большинстве случаев можно просто оставить настройки по умолчанию (см. рис. 5.12).

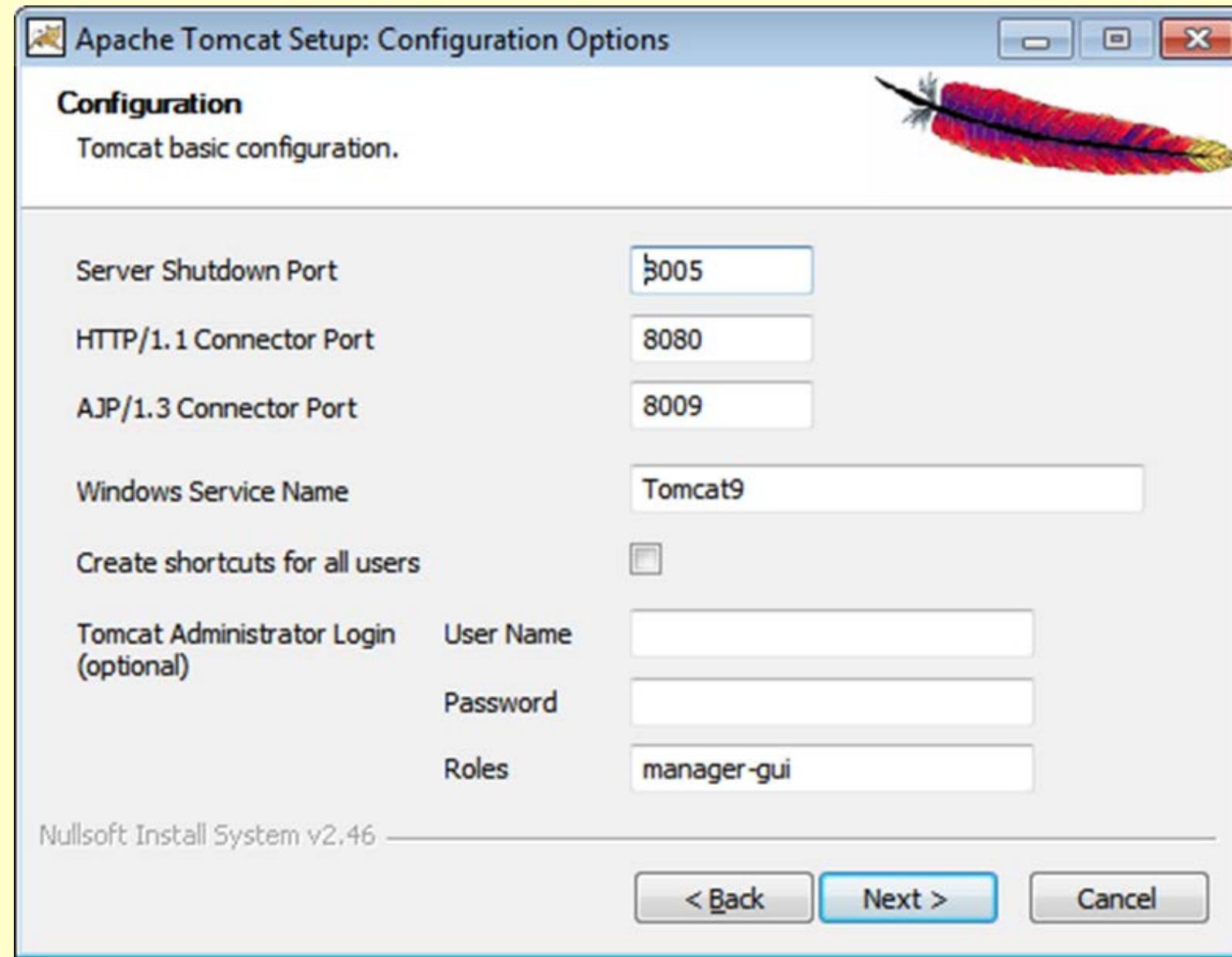


Рис. 5.12

Здесь необходимо указать путь к установленному в системе JRE (см. рис. 5.13).

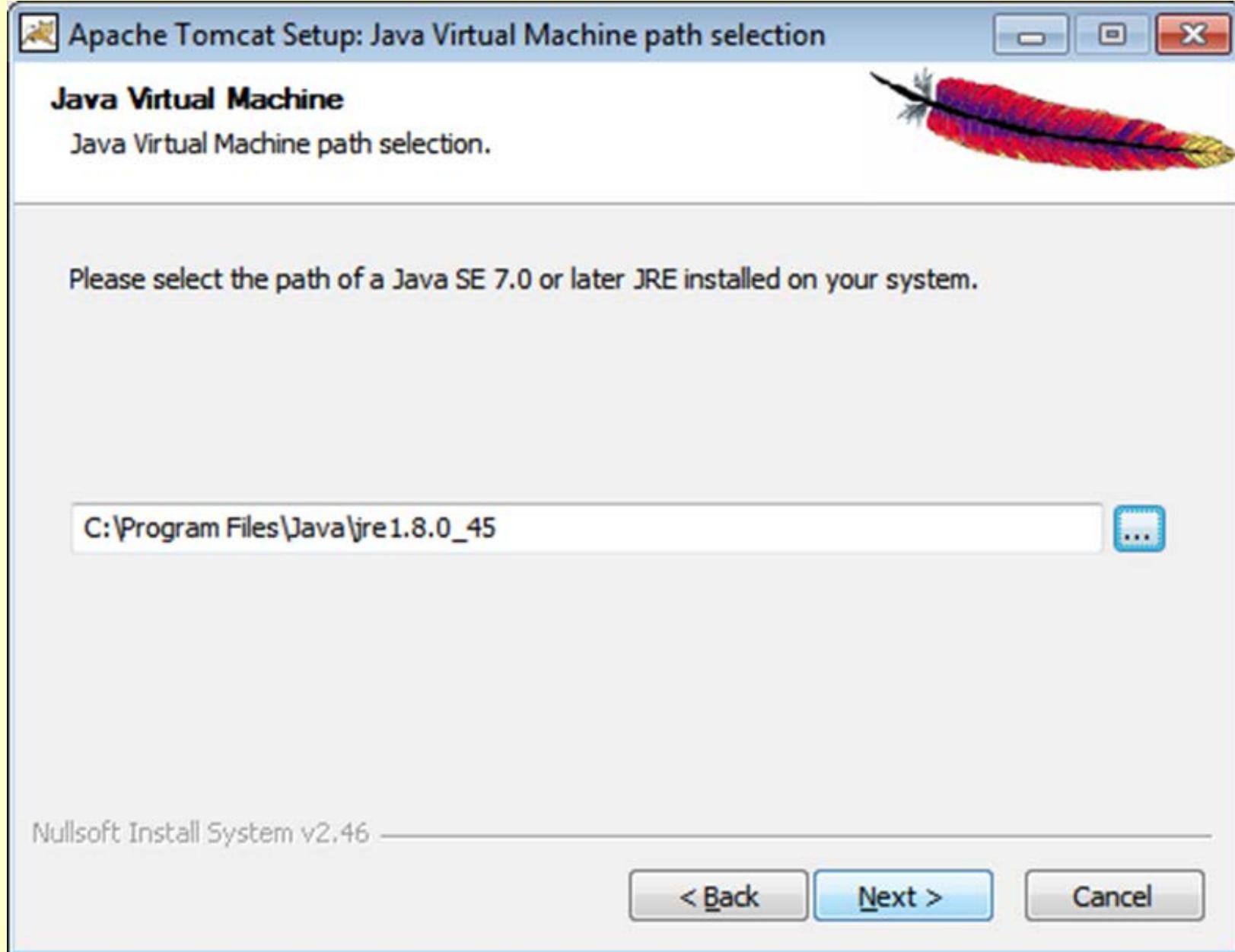


Рис. 5.13.

И наконец, папку установки (см. рис. 5.14).

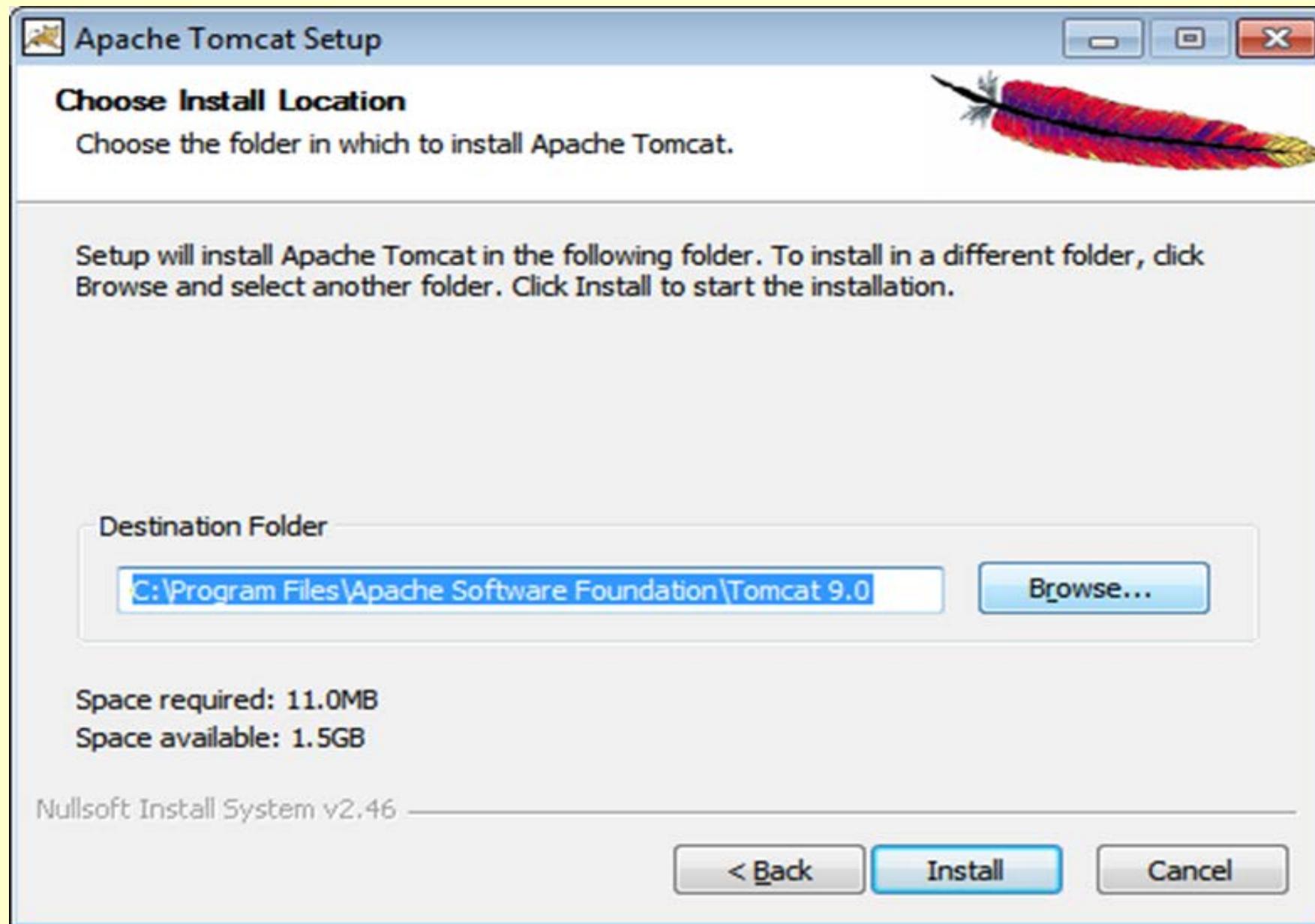


Рис. 5.14

В последнем окне выбираем опцию Run Apache Tomcat и нажимаем Finish, ждем конца процесса установки.

И по окончании установки появится иконка Tomcat.

Установка Tomcat в ОС Linux (Ubuntu)

Чтобы установить Tomcat под Linux нужно запустить терминал и набрать команду:
`sudo apt-get install tomcat7`.

Далее в браузере в адресной строке запускаем <http://localhost:8080> и видим страницу, которую выдает на ваш запрос сервер Tomcat.

Фреймворк Spring и Eclipse

Преимущества фреймворка общеизвестны, однако подробнее о Spring можно узнать на его сайте <http://spring-projects.ru>.

Tomcat также входит в Java-фреймворк Spring.

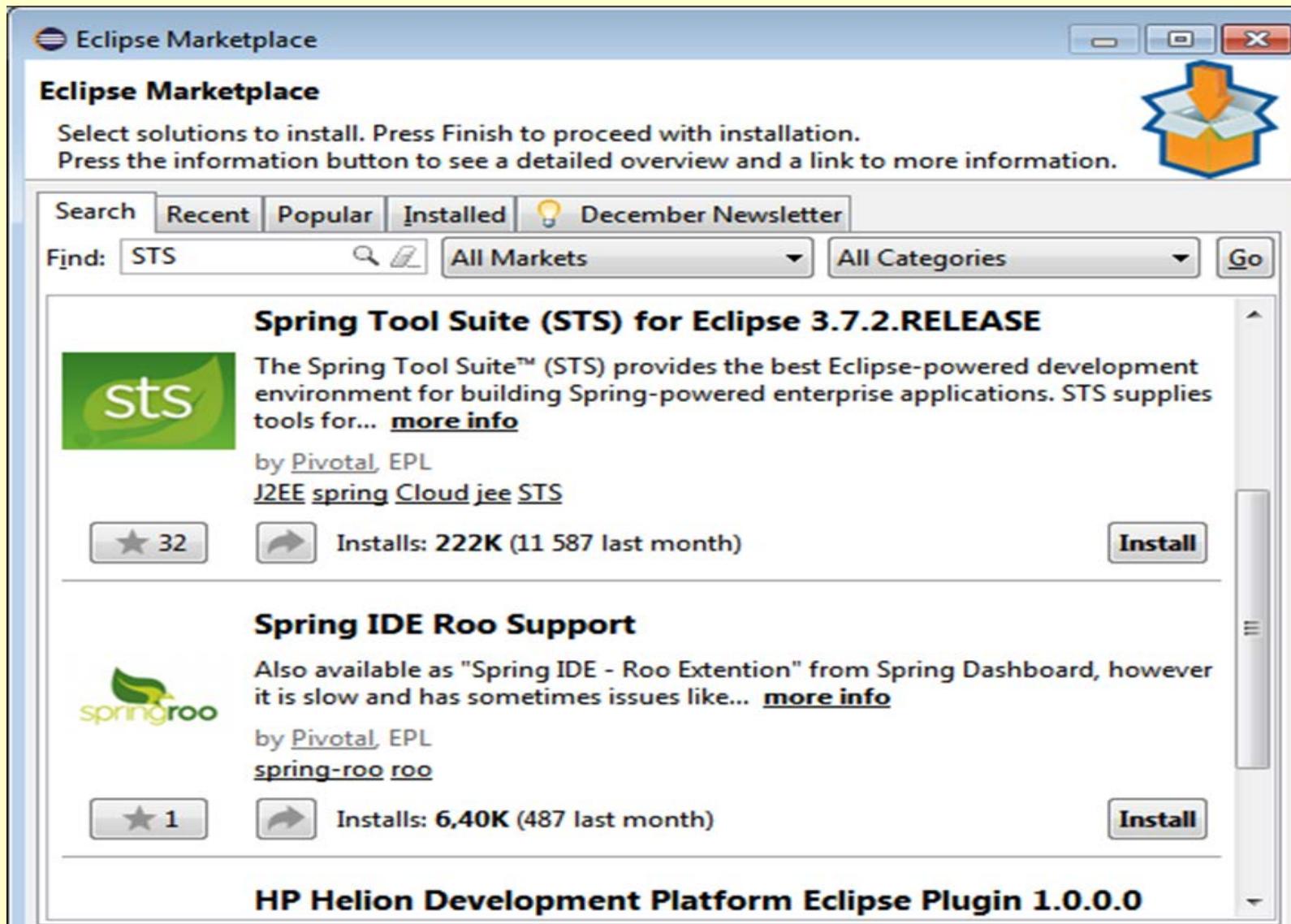
Для разработки веб-приложений используется шаблон Spring MVC.

Однако для того, чтобы развернуть такое приложение от программиста, требуется написать конфигурационный код, причем, как правило, для типовых проектов он одинаков.

Поэтому у начинающих разработчиков в особенности пользуется популярностью инструмент Spring Boot, который автоматизирует и тем самым облегчает этот процесс.

Обратите внимание, что в плагине STS имеется встроенный сервер Tomcat, который запускается, когда в среде разработки запускается на выполнение проект SpringBoot. Таким образом, если на текущем компьютере уже запущен ранее установленный сервер Tomcat с настройками по умолчанию, то произойдет конфликт при попытке открытия на прослушивание серверного сокета на порту 8080. Решением этой проблемы может быть изменение порта на прослушивание при установке сервера, либо остановка ранее установленного сервера, либо можно просто его не устанавливать (удалить).

Spring Boot в Eclipse можно установить с помощью плагина Spring Tools Suite (STS), который легко подключается через Help -> Eclipse Marketplace. Необходимо набрать в строке поиска STS, далее нажать Install на найденном плагине STS (первый на рис. 5.15).



После поиска всех компонентов в окне Confirm Selected Futures необходимо выбрать все и нажать Confirm. В следующем окне нужно принять лицензионное соглашение и нажимать Finish. Теперь все готово к разработке простого клиент-серверного приложения на базе Spring Boot

Рис. 5.15.

Пример

В этом примере продемонстрируем взаимодействие браузера (HTML-страницы) и Java-программы в контейнере веб-сервера по протоколу HTTP. Эта работа должна происходить по следующему алгоритму:

1. Простейшая html-форма, которая с помощью браузера будет отправлять значения двух полей Имя и Фамилия на сервер по кнопке «Отправить». Например:

The screenshot shows a web form with two text input fields and a submit button. The first field is labeled "Фамилия:" and contains the value "Petrov". The second field is labeled "Имя:" and contains the value "Sergey". Below the fields is a blue-outlined button labeled "Отправить" (Send).

Фамилия:	Petrov
Имя:	Sergey
Отправить	

2. Сервер в ответ должен возвратить в браузер одну строку, сформированную из исходных полей, например, «Petrov S.».
3. Эта строка должна появиться в браузере.

Создание html-страницы

Ниже приведен HTML-код простейшей формы (файл 5.4.html):

```
<html>
<form name='test' method='post' action='http://localhost:8080/'>
<p><b>Фамилия:</b><br>
  <input type='text' name='lastname' size='40'>
</p>
<p><b>Имя:</b><br>
  <input type='text' name='firstname' size='40'>
</p>
<p>
  <input type='submit' value='Отправить'>
</p>
</form>
</html>
```

Не вдаваясь в тонкости языка разметки HTML, можно сказать, что тут описана форма с двумя полями ввода и одной кнопкой.

В результате нажатия на кнопку «Отправить» произойдет отправка POST-запроса по URL — <http://localhost:8080/>, откуда затем и будет выведен результат работы серверной части приложения

Создание в Eclipse Spring-проекта

Для выполнения пункта 2 задания необходимо в Eclipse (STS-плагин должен быть установлен) создать проект Spring, воспользовавшись мастером (рис. 5.16).

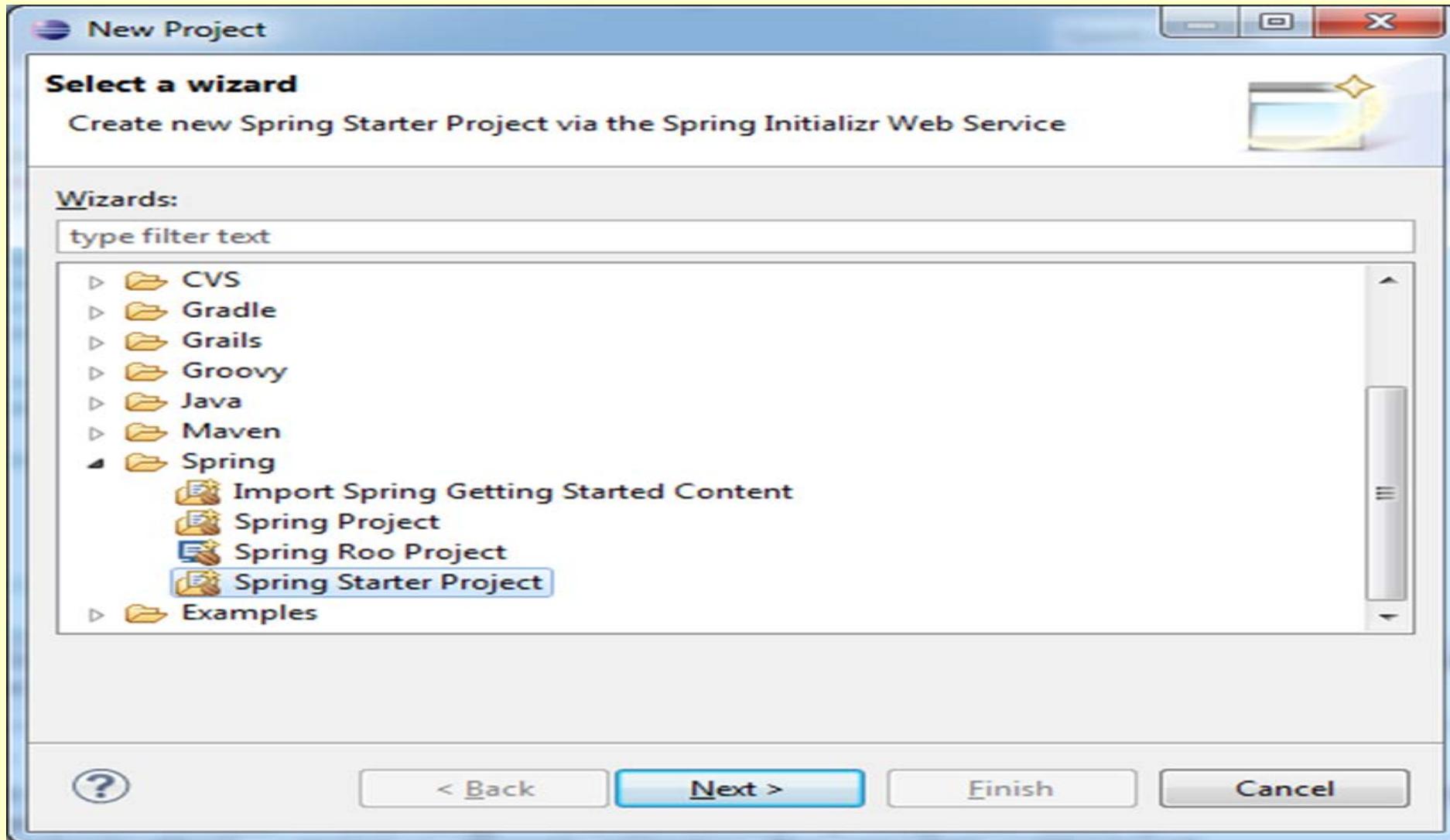


Рис. 5.16.

В открывшемся мастере из полей необходимым для заполнения является только название проекта (name). Остальные можно оставить как есть (см. рис. 5.17).

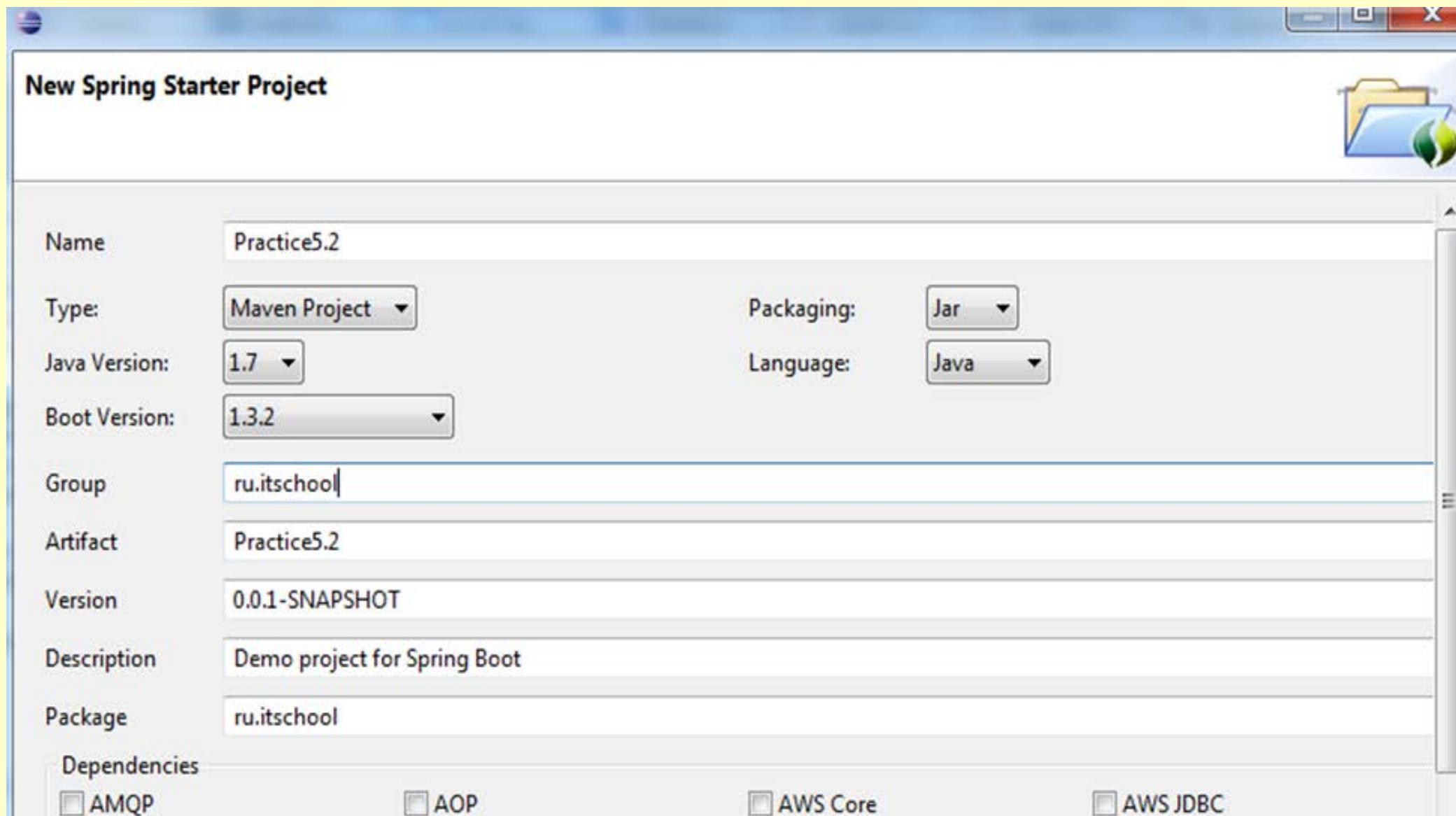


Рис. 5.17.

Однако в нижней части окна Dependencies необходимо установить галочку Web, чтобы в проекте появились необходимые зависимости, подключающие библиотеки для обработки POST и GET-запросов (рис. 5.18).

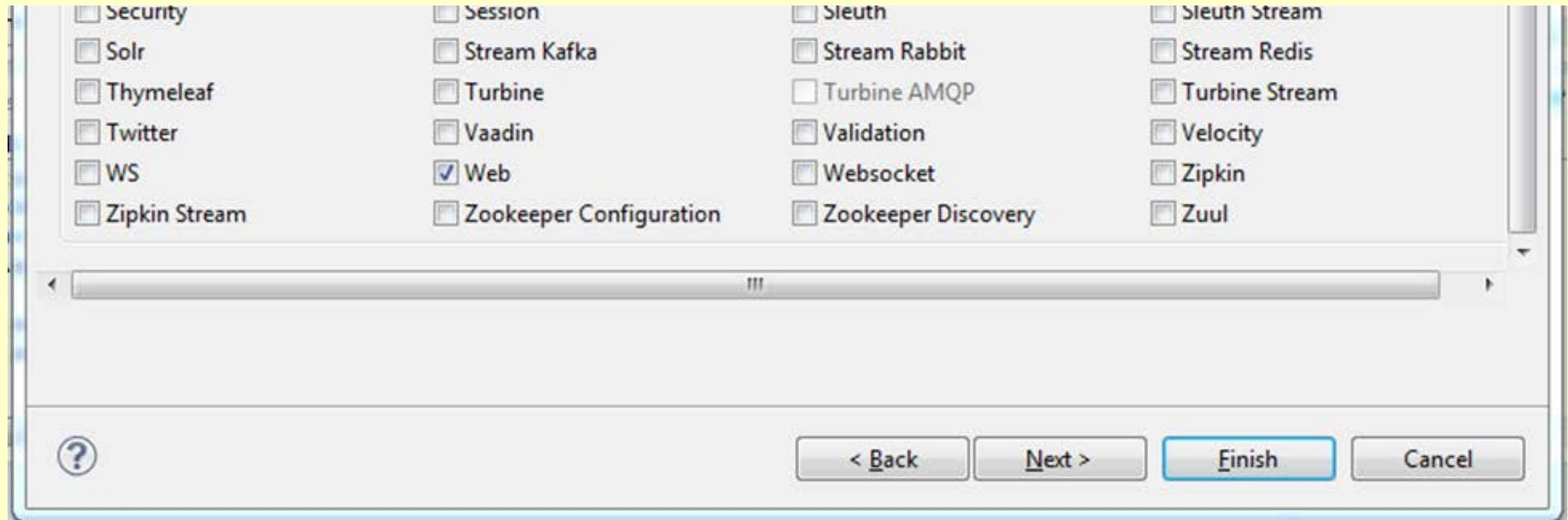


Рис. 5.18.

Далее необходимо нажать кнопку Finish, после чего проект будет полностью сформирован и открыт.

Обработка запросов на сервере

В созданном приложении автоматически был сгенерирован главный

В созданном приложении автоматически был сгенерирован главный класс Application с кодом запуска сервера.

В нем аннотация @SpringBootApplication позволяет Spring Boot сообщить о необходимости выполнения ряда действий для генерации другого кода, конфигураций и т. д.

Для запуска Spring-приложения в имеющемся методе main вызывается метод SpringApplication.run(), который автоматически формирует веб-страницу из имеющихся в приложении классов:

```
package ru.itschool;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Создадим класс для получения из GET-запроса переданной информации (фамилия, имя) и обратного ответа сервера:

```
package ru.samsung.itschool;  
import javax.servlet.http.HttpServlet;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
public class HttpControllerREST extends HttpServlet {  
  
    @RequestMapping("/")  
    public String index(HttpServletRequest request, HttpServletResponse response) {  
        if (request.getParameter("lastname") != null || request.getParameter("firstname") != null) {  
            if (!request.getParameter("lastname").equals("") && !request.getParameter("firstname").equals("")) {  
                String lastname = request.getParameter("lastname");  
                String firstname = request.getParameter("firstname");  
                firstname = firstname.substring(0, 1); //Первая буква  
                return lastname + " " + firstname + ".;"  
            } else  
                return "No POST data lastname or firstname";  
        }  
        return "Not POST data ";  
    }  
}
```

Аннотация @RestController указывает на то, что класс готов принимать веб-запросы, используя Spring MVC. @RequestMapping("/") указывает, по какому адресу (URI) будет доступна страница.

При запуске приложения (ПКМ -> Run As -> SpringBootApp) встроенный сервер Tomcat также будет запущен и будет доступен по адресу localhost на порту 8080.

Теперь можно проверить работу примера, открыв веб-форму и заполнив, нажать кнопку «Отправить».

Реализация сервера на PHP

PHP (англ. PHP: Hypertext Preprocessor — «PHP: препроцессор гипертекста»; произносится пи-эйч-пи) — скриптовый язык общего назначения, интенсивно применяемый для разработки веб-приложений.

В настоящее время поддерживается подавляющим большинством хостинг-провайдеров и является одним из лидеров среди языков, применяющихся для создания динамических веб-сайтов.

PHP — язык программирования, исполняемый на стороне веб-сервера, спроектированный Расмусом Лердорфом (Rasmus Lerdorf).

Язык оказался достаточно гибким и мощным, поэтому приобрел большую популярность и используется в проектах любого масштаба: от простого блога до крупнейших веб-приложений в интернете.

Он получил широкое распространение, благодаря множеству преимуществ, начиная с того, что является свободно распространяемым под особой лицензией (PHP license), легок в освоении, имеет развитую поддержку баз данных, библиотеки и расширения языка, может быть развернут почти на любом сервере, на широком спектре аппаратных платформ и операционных систем.

Среди недостатков PHP разработчики отмечают несогласованность его синтаксиса, он не подходит для создания десктопных приложений или системных компонентов и, самое главное, сложилось мнение, что веб-приложения, написанные на PHP, зачастую имеют проблемы с безопасностью.

Поэтому во многих крупных компаниях, банках существует политика запрета на использование PHP.

Однако, это не мешает его широчайшему распространению на миллионах веб-серверов. К крупнейшим сайтам, использующим PHP, относятся Facebook, Wikipedia и др.

Наиболее популярный PHP сервер — это HTTP сервер Apache. Apache — свободно распространяемый кроссплатформенный сервер, полностью разрабатываемый Apache Software Foundation.

Его основными достоинствами считаются надежность и гибкость конфигурации, поддерживает IPv6.

Помимо PHP Apache поддерживает языки программирования: Python, Ruby, Perl, ASP, Tcl. По некоторым данным Apache установлен на более чем 60% серверах в мире.

Как установить Apache и PHP5?

В ОС Windows этот процесс подробно описан в статье здесь: <http://www.q2w3.ru/2011/01/12/3093/>

Для установки Apache и PHP5 в ОС Linux (Ubuntu) можно воспользоваться руководством отсюда:
<http://help.ubuntu.ru/wiki/apachemysqlphp>

Функциональность, аналогичная ранее рассмотренной на Java-сервере Tomcat, может быть реализована с помощью кода:

```
<?php  
if (isset($_POST["lastname"]) || isset($_POST["firstname"]))  
    if ($_POST["lastname"] != "" && $_POST["firstname"] != "") {  
        $lastname = $_POST["lastname"];  
        $firstname = $_POST["firstname"];  
        $firstname = iconv('UTF-8', 'windows-1251', $firstname); // перевод кодировки нужен для того, чтобы корректно русские  
        // буквы отображались  
        $firstname = substr($firstname, 0, 1);  
        $firstname = iconv('windows-1251', 'UTF-8', $firstname); // перевод кодировки нужен для того, чтобы корректно русские  
        // буквы отображались  
        echo $lastname . " " . $firstname . ".;";  
    } else  
        echo "No POST data lastname or firstname";  
    else  
        echo "Not POST data";  
?>
```

Лекция

Клиент-серверная архитектура мобильных приложений

Общие понятия

Архитектура клиент-сервер предполагает наличие трех компонент:

- клиентской части (пользователь);
- серверной части (удаленный сервис);
- среды коммуникации.

В мобильных приложениях клиентская часть располагается на мобильном устройстве и, как правило, предназначена для взаимодействия с пользователем устройства.

Серверная часть располагается либо на отдельном компьютере, либо на другом мобильном устройстве.

Сервер предоставляет клиенту некий сервис и напрямую с пользователем не взаимодействует.

Для связи между клиентской и серверной частью используется коммуникационная среда, в случае мобильных приложений, чаще всего, обеспечиваемая протоколами интернет.

Одним из популярных интернет-протоколов является Hyper Text Transmission Protocol (HTTP).

Если серверная часть приложения базируется на системе обработки HTTP, то говорят о мобильном HTTP-приложении.

В этом случае все коммуникации между клиентом и сервером организуются с использованием стандартных HTTP-запросов и ответов, рассмотренных в главе 5.2 (см. рис. 5.19).

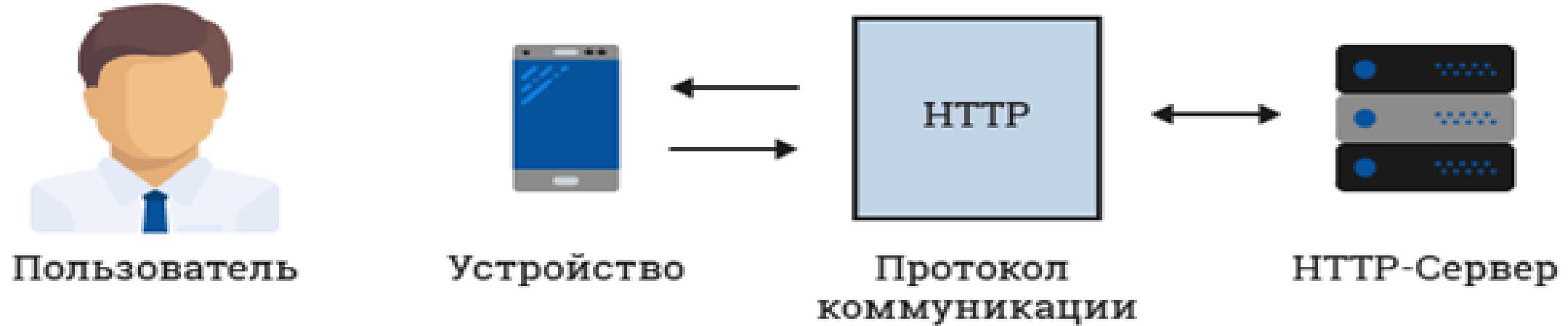


Рис. 5.19

Запросы от мобильного клиента к серверу могут быть как синхронными, так и асинхронными.

Понятия синхронных и асинхронных сообщений по своей сути сходны с понятиями синхронных и асинхронных процессов, рассмотренных в модуле 3. При использовании синхронных сообщений работа приложения блокируется до тех пор, пока не получен ответ от сервера.

Если используются асинхронные сообщения, то клиент продолжает работу в штатном режиме, не дожидаясь ответа.

Отправка запросов из Android-приложений

Чтобы отправлять запросы через коммуникационную среду интернет, Android-приложению должен быть разрешен доступ в интернет.

Для этого необходимо добавить в манифест строку:

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

Для написания HTTP-приложений под Android существует ряд классов, которые позволяют быстро реализовывать базовую функциональность.

Они содержатся в пакете org.apache.http и его подпакетах.

Интерфейс клиента называется HttpClient. Экземпляр класса с этим интерфейсом легче всего создать, используя класс DefaultHttpClient.

В случае вызова new DefaultHttpClient() создается стандартный клиент.

Apache HTTP клиент был встроен в Android API до версии 22. Начиная с версии 23 (Marshmallow) эта библиотека была полностью удалена.

Таким образом, для работы с HTTP нужно воспользоваться одним из следующих способов:

- при создании проекта указать API 22, или
- если вы указываете SDK выше чем 22, то можно разрешить использование отключенной библиотеки указав строку `useLibrary 'org.apache.http.legacy'` в файле `build.gradle (app)` в разделе `android`, или
- отказаться от использования указанной библиотеки и использовать вместо нее другую, например OkHTTP

```
HttpClient httpclient = new DefaultHttpClient();
```

Чтобы отправлять и получать данные, клиент использует объекты, имплементирующие интерфейс `HttpUriRequest`.

Для Android разработаны классы, приспособленные для частных видов запросов. Например, для отправки данных на сервер используется `HttpPost`, для получения данных с сервера — `HttpGet`.

Объект `HttpPost` можно создать следующим образом:

```
HttpPost httppost = new HttpPost("http://192.168.72.3:8080");
```

В конструктор передается URI-сервера (в виде объекта класса `URI` или класса `String`).

Это может быть как IP-адрес, с указанием порта или без него, так и символьное доменное имя.

Существуют и другие виды HTTP-запросов, для которых разработаны специальные классы -`HttpDelete`, `HttpPut`, `HttpOptions` и т. д.

Их особенности и варианты использования описаны в документации к пакету `org.apache.http.client.methods`.

Выполнение запроса осуществляется с помощью метода `execute()` интерфейса `HttpClient`.

Существуют различные имплементации метода с различными входными параметрами.

Если в качестве аргумента используется объект одного из классов HTTP-запросов.

Метод может бросать исключения `IOException` и `ClientProtocolException`.

Первое вырабатывается в случае, если возникли проблемы с соединением, второе — если вернулась ошибка HTTP-протокола.

Метод возвращает ответ сервера в виде объекта, имплементирующего интерфейс `HttpResponse`.

Поэтому конструкция по отсылке запроса на сервер и получения ответа должна быть такой:

```
HttpResponse response = httpclient.execute(httppost);
```

Для извлечения полезной информации из объекта, имплементирующего `HttpResponse`, используются интерфейс `HttpEntity` и класс `EntityUtils`.

Первый из них описывает блок пользовательских данных, содержащийся в HTTP-сообщении.

Используя метод интерфейса `HttpResponse getEntity()`, можно получить эту часть сообщения.

```
HttpEntity entity = response.getEntity();
```

Класс `EntityUtils` — это небольшая библиотека статических методов для обработки блоков пользовательских данных.

Там есть функции преобразования данных в строки и массив байтов.

Вызов метода из этого класса таков:

```
String answerHTTP = EntityUtils.toString(entity);
```

Пример

Разработаем программу для Android, которая реализует задание, аналогичное показанному в предыдущем примере.

При реализации асинхронного клиент-серверного приложения важно все коммуникационные операции инкапсулировать в класс, расширяющий класс AsyncTask, как это было показано в модуле 3.

В данном упражнении достаточно переопределить методы doInBackground() и onPostExecute(), но для решения других задач можно использовать весь арсенал средств, предоставляемых AsyncTask.

Итак, имплементация клиентской части требуемого приложения будет выглядеть следующим образом:

```
package com.samsung.itschool.app5_3_1;
```

```
import android.os.AsyncTask;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.HttpClient;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;
import org.apache.http.util.EntityUtils;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
```

```
public class MainActivity extends AppCompatActivity {  
    TextView lastnameF;  
    String answerHTTP;  
    String lastnameS, firstnameS;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        lastnameF = (TextView) findViewById(R.id.lastnameF);  
    }  
  
    public void sendPOST(View view) {  
        EditText lastname = (EditText) findViewById(R.id.lastname);  
        EditText firstname = (EditText) findViewById(R.id.firstname);  
        lastnameS = lastname.getText().toString();  
        firstnameS = firstname.getText().toString();  
        new MyAsyncTask().execute("");  
    }  
}
```

```
class MyAsyncTask extends  
AsyncTask<String, String, String> {  
  
    @Override  
    protected String doInBackground(String...  
params) {  
        // Создаем HttpClient и Post Header  
        HttpClient httpclient = new  
DefaultHttpClient();  
        HttpPost httppost = new  
HttpPost("http://192.168.72.3:8080");  
        try {  
            List<NameValuePair> nameValuePairs  
= new ArrayList<NameValuePair>(2);  
            nameValuePairs.add(new  
BasicNameValuePair("lastname",  
lastnameS));  
            nameValuePairs.add(new  
BasicNameValuePair("firstname",  
firstnameS));  
            httppost.setEntity(new  
UrlEncodedFormEntity(nameValuePairs,  
"UTF-8"));  
            // Выполняем HTTP Post запрос  
            HttpResponse response =  
httpclient.execute(httppost);  
            if (response.getStatusLine().getStatusCode()  
== 200) {  
                HttpEntity entity = response.getEntity();  
                answerHTTP = EntityUtils.toString(entity);  
            }  
        } catch (ClientProtocolException e) {  
            // TODO Auto-generated catch block  
        } catch (IOException e) {  
            // TODO Auto-generated catch block  
        }  
        return null;  
    }  
  
    @Override  
    protected void onPostExecute(String result) {  
        super.onPostExecute(result);  
        lastnameF.setText(answerHTTP);  
    }  
}
```

Пользовательский интерфейс должен включать:

- два поля текстового редактирования lastname, firstname;
- кнопку «Отправить» с обработчиком под именем sendPOST.

Для корректной работы серверная часть должна быть несколько иной, чем в предыдущем примере. Класс HttpControllerREST изменится следующим образом:

```
import javax.servlet.http.HttpServlet;  
  
import javax.servlet.http.HttpServletRequest;  
  
import javax.servlet.http.HttpServletResponse;  
  
import org.springframework.web.bind.annotation.RequestMapping;  
  
import org.springframework.web.bind.annotation.RestController;
```

```
@RestController
public class HttpControllerREST extends HttpServlet {
    @RequestMapping("/")
    public String index(HttpServletRequest request, HttpServletResponse response) {
        if (request.getParameter("lastname") != null || request.getParameter("firstname") != null)
            if (!request.getParameter("lastname").equals("") && !request.getParameter("firstname").equals(""))
                String lastname = request.getParameter("lastname");
                String firstname = request.getParameter("firstname");
                firstname = firstname.substring(0, 1);
                return lastname + " " + firstname + ".";
        } else
            return "No POST data lastname or firstname";
        return "<h1>Greetings from Spring Boot!</h1>" +
            + "<form name='test' method='post' action=>" +
            + "<p><b>Фамилия:</b><br>" +
            + "<input type='text' name='lastname' size='40'></p>" +
            + "<p><b>Имя:</b><br>" +
            + "<input type='text' name='firstname' size='40'></p>" +
            + "<p><input type='submit' value='Отправить'></p>" + " </form>";
    }
}
```

Обратите внимание, что посылаемая через интернет пользовательская информация разделена на пары «ключ — значение».

Ключи в упражнении — `firstname` и `lastname`.

Они одинаковы как для клиентской части приложения (хранится в `nameValuePairs`), так и для серверной части.

По этим ключам происходит присвоение и извлечение соответствующих значений, которые в дальнейшем используются в программе.

Класс `Application` останется прежним:

```
import java.util.Arrays;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.context.ApplicationContext;
```

```
@SpringBootApplication  
public class Application {  
    public static void main(String[] args) {  
        ApplicationContext ctx = SpringApplication.run(Application.class,args);  
    }  
}
```

Аналогичная реализация сервера на PHP приведена ниже:

```
<?php  
if (isset($_POST["lastname"]) || isset($_POST["firstname"]))  
    if ($_POST["lastname"] != "" && $_POST["firstname"] != "") {  
        $lastname = $_POST["lastname"];  
        $firstname = $_POST["firstname"];  
        $firstname = iconv('UTF-8', 'windows-1251', $firstname); // перевод кодировки нужен для того, чтобы корректно русские буквы отображались  
        $firstname = substr($firstname, 0, 1);  
        $firstname = iconv('windows-1251', 'UTF-8', $firstname); // перевод кодировки нужен для того, чтобы корректно русские буквы отображались  
        echo $lastname . " " . $firstname . ".;"  
    } else  
        echo "No POST data lastname or firstname";  
else  
    echo "  
<form name='test' method='post' action='>  
<p><b>Фамилия:</b><br>  
<input type='text' name='lastname' size='40'></p>  
<p><b>Имя:</b><br>  
<input type='text' name='firstname' size='40'></p>  
<p><input type='submit' value='Отправить'></p>  
</form>";  
?>
```

Форматы JSON и XML. Сериализация

Для передачи структуры данных по коммуникационным протоколам необходимо перед передачей перевести структуру в последовательность битов.

Такое преобразование называется сериализацией.

Для того чтобы принимающая сторона могла эффективно обработать эти данные, существует обратный процесс, называемый десериализацией.

В процессе десериализации происходит восстановление структуры данных в первоначальный вид.

Распространенным видом сериализации является перевод объектов программы в файл.

В этом случае объект заполняется нужными данными, потом вызывается функция сериализации, которая преобразует его в файл некоторого формата. Файл передается через коммуникационную среду.

Принимающая сторона отправляет файл в функцию десериализации.

После обработки получатель располагает исходным объектом, заполненным нужными данными.

Любой из схем сериализации присущ то, что кодирование данных происходит последовательно по определению, и поэтому необходимо считать весь файл и только потом воссоздать исходную структуру данных

Формат JSON

В качестве формата файла сериализации часто используют XML.

Однако возможно использование и других форматов.

В частности, при разработке мобильных приложений очень популярен JSON (JavaScript Object Notation) — специальный текстовый формат обмена данными, основанный на JavaScript.

Несмотря на то что синтаксис описания данных напоминает JS, формат является независимым от языка и может использоваться практически с любым языком программирования.

За счет лаконичности синтаксиса по сравнению с XML, формат JSON является более подходящим для сериализации сложных структур.

Далее представлен пример сериализованного объекта «Пользователь» в обоих форматах.

JSON:

```
{  
  "firstname": "Александр",  
  "lastname": "Викторов",  
  "from": {  
    "region": "Нижегородская область",  
    "town": "Дзержинск",  
    "school": 17  
  },  
  "phone": [  
    "312 123-1234",  
    "312 123-4567"  
  ]  
}
```

XML:

```
<user>
  <firstname>Александр</firstname>
  <lastname>Викторов</lastname>
<from>
  <region> Нижегородская область</region>
  <town>Дзержинск</town>
  <school>17</school>
</from>
<phone>
  <phonenumber>312 123-1234</phonenumber>
  <phonenumber>312 123-4567</phonenumber>
</phone>
</user>
```

В JSON-формате предусмотрено использование двух структур:

- набор пар «ключ — значение». Ключом может быть только строка, значением — любая запись. В разных языках программирования ей соответствуют объект, запись, структура, словарь, хеш, именованный список или ассоциативный массив;
- упорядоченный набор значений. В большинстве языков это реализовано как массив, вектор, список или последовательность.

Названные структуры данных удобны: большая часть алгоритмических языков программирования высокого уровня поддерживает их в той или иной форме.

Поэтому даже если клиентская и серверная части приложения написаны на разных языках, организовать их взаимодействие не представляет труда.

В качестве значений в JSON-формате возможно использование следующих записей:

- объект — это неупорядоченное множество пар «ключ — значение», заключенное в фигурные скобки { }. Ключ и значение разделяются символом «:». Пары «ключ — значение» разделяются запятыми;
- одномерный массив — это упорядоченное множество значений. Массив заключается в квадратные скобки []. Значения ячеек массива разделяются запятыми;
- простое значение может быть строкой в двойных кавычках, числом или одним из литералов: true, false, null.

Записи могут быть вложены друг в друга. В приведенном ранее примере по ключу from находится объект, состоящий из трех пар «ключ — значение».

```
"from": {  
    "region": "Нижегородская область",  
    "town": "Дзержинск",  
    "school": 17  
}
```

Одномерный массив строк задан ключом «phone».

```
"phone": [  
    "312 123-1234",  
    "312 123-4567"  
]
```

Строка в JSON — это строка, состоящая из символов юникода, заключенного в двойные кавычки.

Символы могут быть указаны с использованием escape-последовательностей, начинающихся с обратной косой черты «\», или записаны в кодировке UTF-8.

Для записи чисел используется только десятичный формат.

Пробелы могут быть вставлены между любыми двумя синтаксическими элементами.

Помимо описанного среди основных отличий от XML:

- JSON не поддерживает многострочные тексты — однако можно в рамках одной строки использовать escape-последовательность «\n» вместо переводов строк. Например, { «greeting» : «Поздравляю
- с новым годом!\nЖелаю счастья!» };
- Если текст в JSON содержит спецсимвол, то его необходимо вручную экранировать символом escape-последовательности: { «title» : «\«Rock&roll\» = life» };
- В XML нет синтаксического представления массивов и списков, тогда как в JSON есть.

Это, конечно, не означает, что в XML нельзя сериализовать список — конечно, можно.

В этом случае программист просто имеет в виду, что группа узлов с одинаковым названием и есть элементы списка.

Сериализация с помощью класса Gson

Сериализация в JSON возможна разными способами.

Рассмотрим один из простейших способов — с помощью класса Gson из пакета com.google.gson.

Предположим, мы хотим сериализовать объект класса User.

```
public class User {  
    public String firstname; // имя  
    public String lastname; // фамилия  
    public int school; // номер школы  
}
```

Следующий код сериализует объект класса User в JSON (код для Android).

```
User student = new User();  
student.firstname = "Александр";  
student.lastname = "Викторов";  
student.school = 17;  
Gson gson = new Gson();  
Log.i("GSON", gson.toJson(student));
```

После исполнения программы в логах появится строка:

```
{"firstname": "Александр", "lastname": "Викторов", "school": 17}
```

Чтобы производить десериализацию в Gson есть другая функция.

Предположим, из JSON-строки jsonText необходимо построить объект для работы в приложении. Тогда код будет выглядеть так:

```
// строка JSON - {"firstname": "Александр", "lastname": "Викторов", "school": 17}

String jsonText
= "{\"firstname\": \"Alexandr\", \"lastname\": \"Viktorov\", \"school\": 17}";

Gson gson = new Gson();

User user = gson.fromJson(jsonText, User.class);

Log.i("GSON", "Student: " + user.firstname + " " + user.lastname + ", school N " +
user.school);
```

После выполнения кода в логах появится строка со значениями из структуры данных User.

Таким образом происходит сериализация/десериализация Java-объектов в JSON-строки.

Примерно аналогичным образом выполняется и сериализация/десериализация Java-объектов в XML-строки.

Однако рассмотренный способ сериализации — это лишь частный случай более общего процесса сериализации.

Следует помнить, что указанные библиотеки — это надстройки над механизмом сериализации в Java.

В общем случае сериализация в Java — это возможность представления любого объекта в виде простой последовательности байт («серии»).

В этом случае появляется возможность передачи объектов через любые потоки. Больше можно прочесть здесь:

- <https://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html>
- <http://info.javarush.ru/translation/2013/09/03/Как-работает-сериализация-в-Java.html>
- <http://www.javable.com/tutorials/fesunov/lesson17/>

Для реализации клиент-серверных приложений, коммуницирующих по протоколу HTTP, удобно использовать специально разработанные для таких целей пакеты классов, или как их часто называют библиотеки.

Одной из наиболее популярных библиотек является Retrofit. К ее достоинствам относят:

- нет нужды делать запросы к HTTP API в отдельном потоке;
- сокращается длина кода и, соответственно, ускоряется разработка;
- возможно подключение стандартных пакетов для конвертации JSON в объекты и обратно (например, пакета Gson);
- динамическое построение запросов;
- обработка ошибок;
- упрощенная передача файлов.

Логика работы библиотеки основывается на аннотациях.

Благодаря их использованию можно описывать динамические запросы на сервер.

Для описания запросов к серверу необходимо объявить интерфейс, который впоследствии будет использоваться при генерации запросов.

Перед каждым методом интерфейса должна стоять аннотация, основываясь на которой, Retrofit определяет, какого типа запрос обрабатывается данным методом. Также с помощью аннотаций можно указывать параметры запроса.

Вот так, например, выглядит описание GET-запроса:

```
import retrofit.client.Response;  
import retrofit.http.GET;
```

```
public interface UserService {  
    @GET("/greeting/user")  
    Response fetchUser();  
}
```

Адрес сайта, которым отправляется запрос, не указывается.

Он будет передан на этапе создания соединения клиент-сервер.

Аннотация содержит лишь путь к PHP-файлу на сервере.

В классе Response содержится информация о статусе запроса и ответ от сервера.

POST-запрос выглядит схоже с GET:

```
import retrofit.client.Response;  
import retrofit.http.POST;
```

```
public interface UserService{  
    @POST("/greeting/registration")  
    Response registerUser();  
}
```

Можно изменять статический маппинг URI на динамический:

```
@GET("/greeting/{firstName}/{lastName}")
```

```
Response fetchUser(@Path("firstName") String firstName, @Path("lastName")  
String lastName);
```

В этом случае Retrofit заменит {firstName} и {lastName} на фактические части URL, которые и будут переданы методу при вызове. Сами аргументы метода должен быть аннотированы Path, а в скобках должно заключаться обозначение конкретной части URI.

Для того чтобы задать запросу параметры, используется аннотация @Query. Слово, указанное в скобках рядом с аннотацией, будет ключом, а аннотированный аргумент значением.

```
@GET("/greeting/user")
```

```
Response fetchUser(@Query("name") String name);
```

Создание соединения клиент-сервер выглядит, как серия вызовов методов библиотеки Retrofit. Например, так:

```
Retrofit retrofit = new Retrofit.Builder()  
    .baseUrl("http://192.168.72.3")  
    .addConverterFactory(GsonConverterFactory.create())  
    .build();  
  
UserService service = retrofit.create(UserService.class);
```

Через параметр метода `baseUrl()` передается адрес сервера.

`GsonConverterFactory` — это класс для конвертации объектов в JSON.

В нем используется уже рассмотренный нами класс `Gson`. `UserService` — интерфейс с запросами HTTP API.

После того, как соединение создано, можно его эксплуатировать, вызывая методы, привязанные к HTTP API.

```
Response resp = service.fetchUser(firstname, lastname);
```

Если ожидается ответ в виде специально созданной структуры данных, то используется конструкция Call<T>, где T — тип возвращаемой структуры данных. Например, интерфейс и структура могут выглядеть так:

```
public interface UserService {  
    @GET("/greeting/user")  
    Call<User> fetchUser();  
}  
  
public class User {  
    public String firstName;  
    public String lastName;  
    public String fullName;  
}
```

Call — это класс, который знает, что такое десериализация и может работать с HTTP-запросами.

Один экземпляр этого класса — это одна пара «запрос — ответ». Экземпляр может быть использован лишь единожды.

Чтобы использовать ту же пару повторно, необходимо использовать метод clone().

Вот как выглядит запрос, в ответ на который ожидается получить специальную структуру данных.

```
Call<User> call = service.fetchUser(firstname, lastname);
```

```
try {
```

```
    Response<User> userResponse = call.execute();
```

```
    userFromServer = userResponse.body();
```

```
} catch (IOException e) {
```

```
    e.printStackTrace();
```

```
}
```

Как видно, сначала создается вызов запроса, и лишь потом он выполняется методом execute().

Это дает возможность передавать созданный запрос в другие классы и выполнять его в любом месте программы.

Если попытаться выполнить вызов запроса дважды, то будет брошено исключение IllegalStateException.

```
userResponse = call.execute();
```

// это вызовет IllegalStateException:

```
userResponse = call.execute();
```

```
Call<User> call2 = call.clone();
```

// А это нет:

```
userResponse = call2.execute();
```

Если вызов клонировать и выполнить уже объект-клон, то программа сработает без ошибок.

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.



Widgit Symbols - Virtual Box

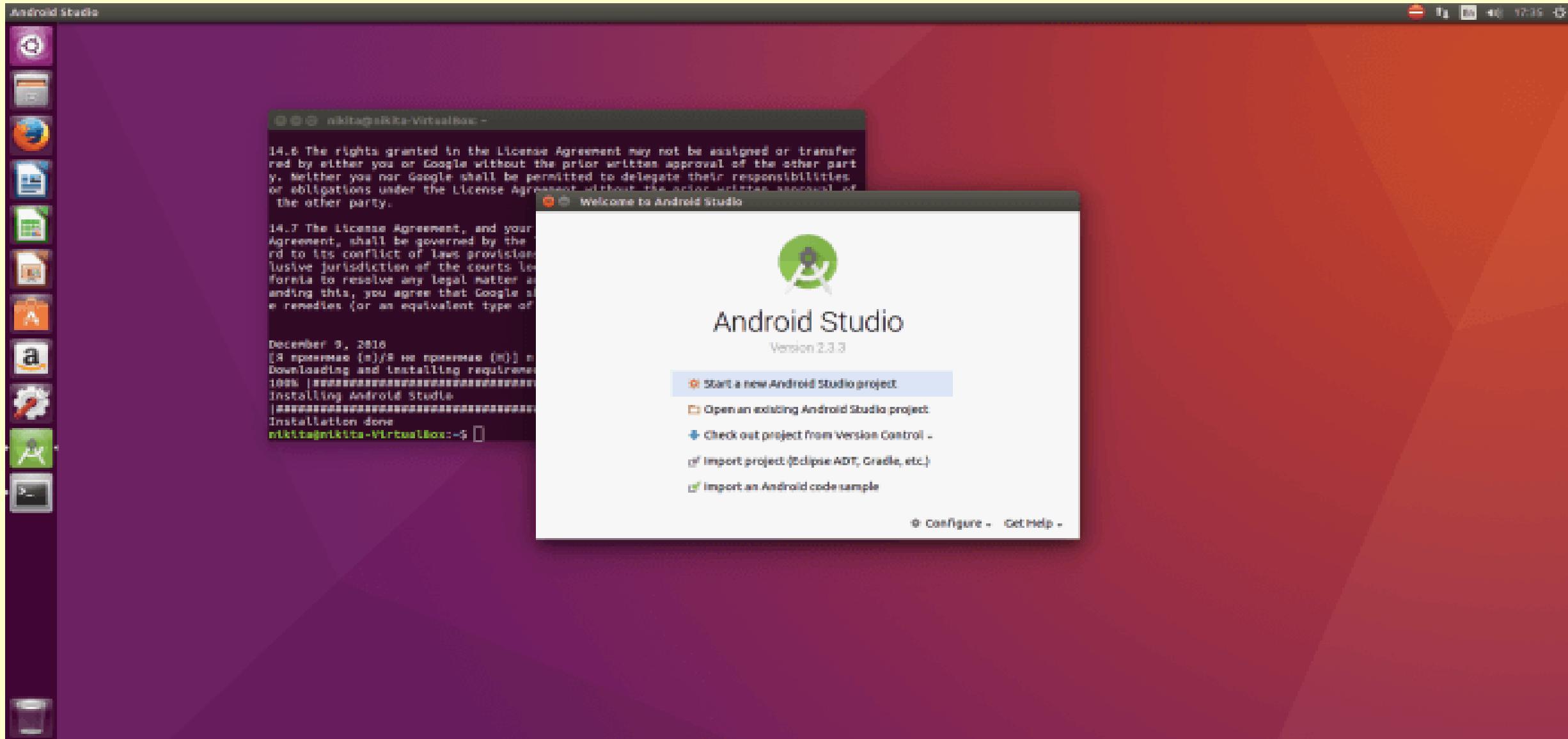
14.5 EXPORT RESTRICTIONS. THE SDR IS SUBJECT TO UNITED STATES EXPORT LAWS AND REGULATIONS. YOU MUST COMPLY WITH ALL DOMESTIC AND INTERNATIONAL EXPORT LAWS AND REGULATIONS THAT APPLY TO THE SDR. THESE LAWS INCLUDE RESTRICTIONS ON DESTINATIONS, END USERS AND END USE.

14.8 The rights granted in the License Agreement may not be assigned or transferred by either you or Google without the prior written approval of the other party. Neither you nor Google shall be permitted to delegate their responsibilities or obligations under the License Agreement without the prior written approval of the other party.

14.7 The License Agreement, and your relationship with Google under the License Agreement, shall be governed by the law of the State of California without regard to the conflict of laws provisions. You and Google agree to submit to the exclusive jurisdiction of the courts located within the county of Santa Clara, California to resolve any legal matter arising from the License Agreement. Notwithstanding this, you agree that Google shall still be allowed to apply for injunctive remedies (or an equivalent type of urgent legal relief) in any jurisdiction.

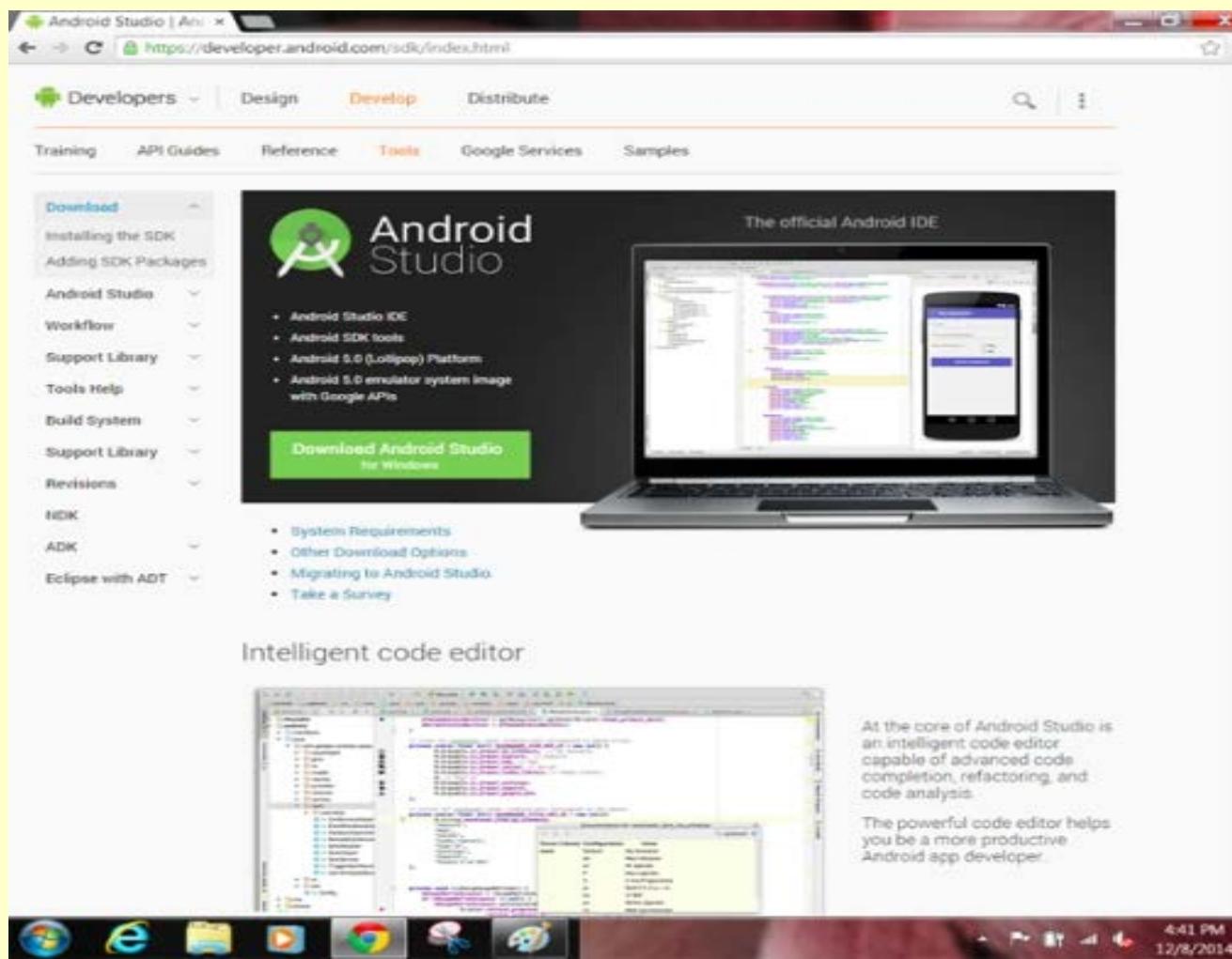
```
December 9, 2016  
(% spackman (v))@% spackman (v)) ~  
Downloading and installing requirements  
45%
```

После того, как будет установлен SDK, можно начинать пользоваться Android Studio.

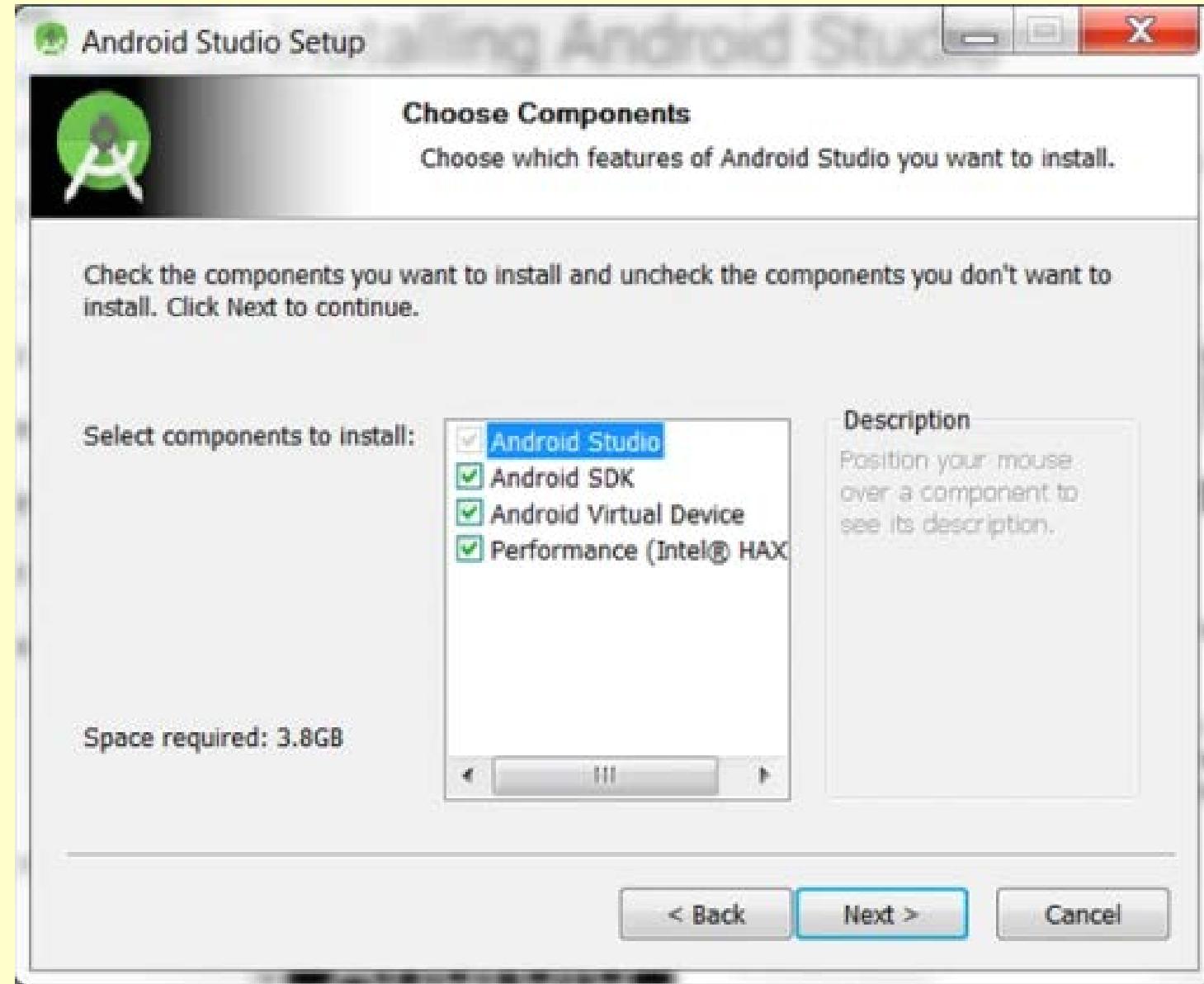


Инсталляция Android Studio под Windows

Перейдите по этой [ссылке](https://developer.android.com/sdk/index.html) на сайт Андроид-разработчиков, чтобы инсталлировать Android Studio. Эта страница определит Вашу операционную систему автоматически.

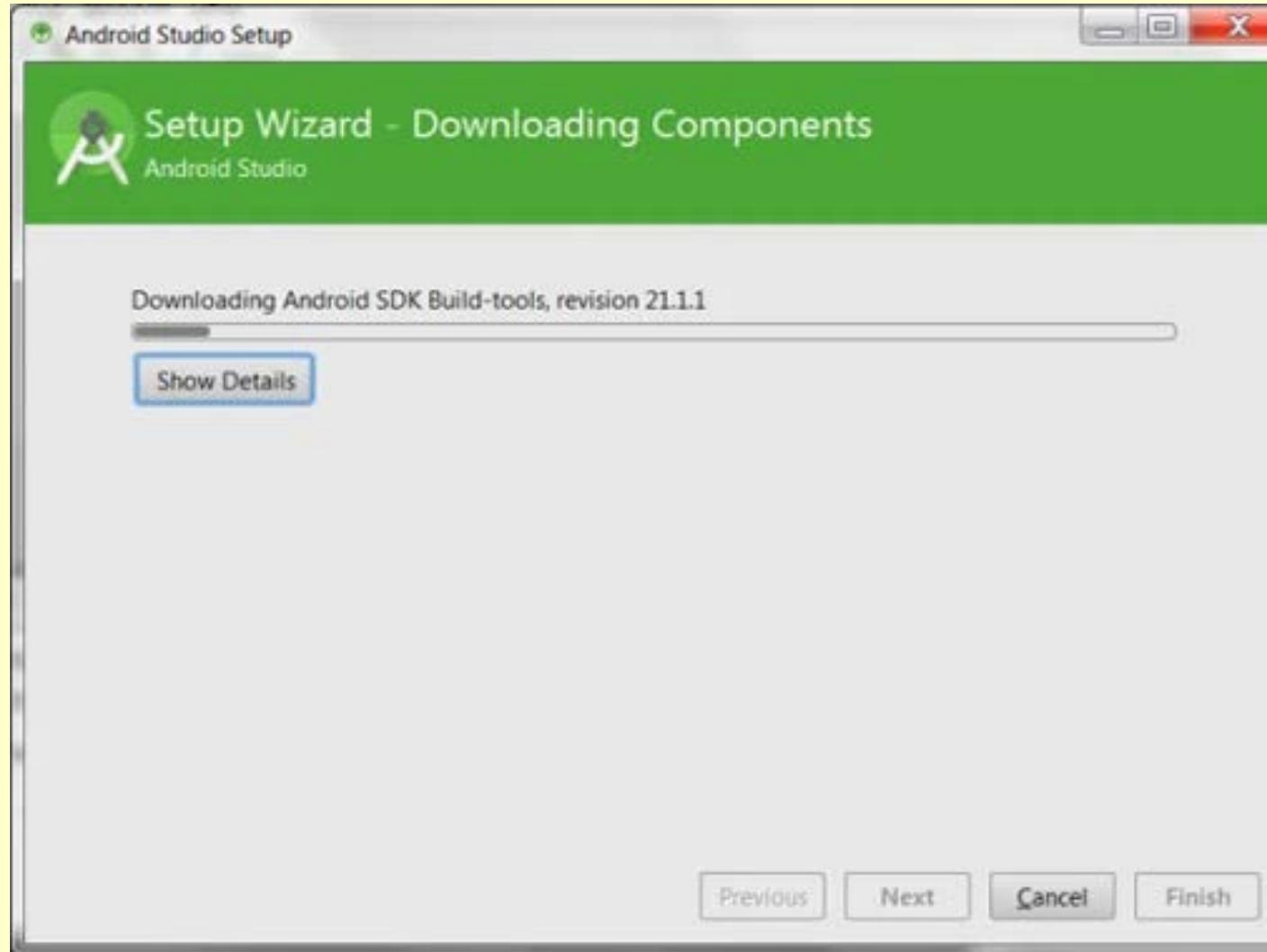


Когда доберетесь до этого экрана, убедитесь, что все компоненты выбраны.



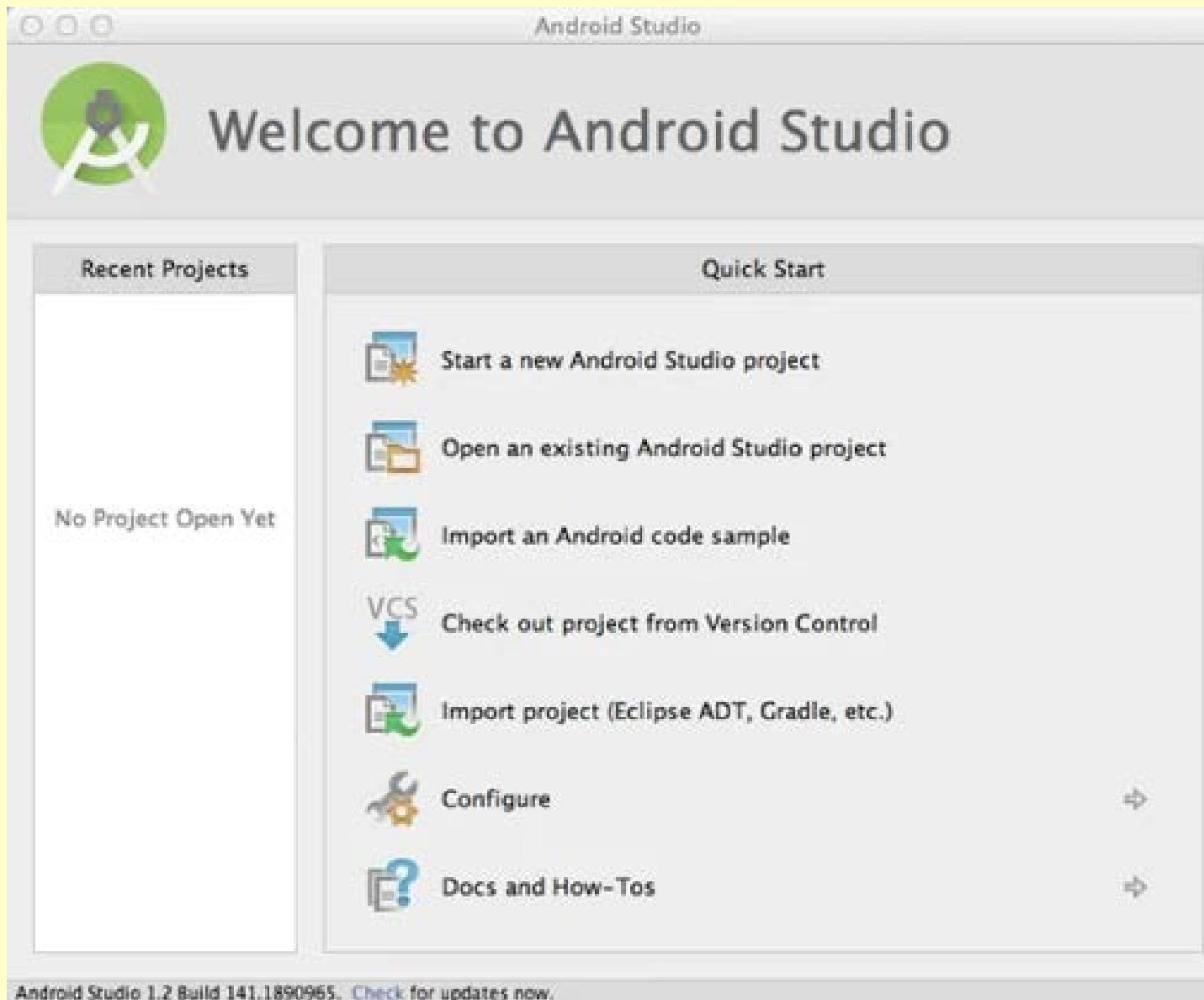
Мастер установки

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:

jdk-14+36
AdoptOpenJDK [latest](#)

Windows
2008r2 or later

x64

Normal

Checksum
(SHA256)



JDK - 189 MB

Checksum
(SHA256)



JDK - 212 MB

Checksum
(SHA256)



JRE - 40 MB

Checksum

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.

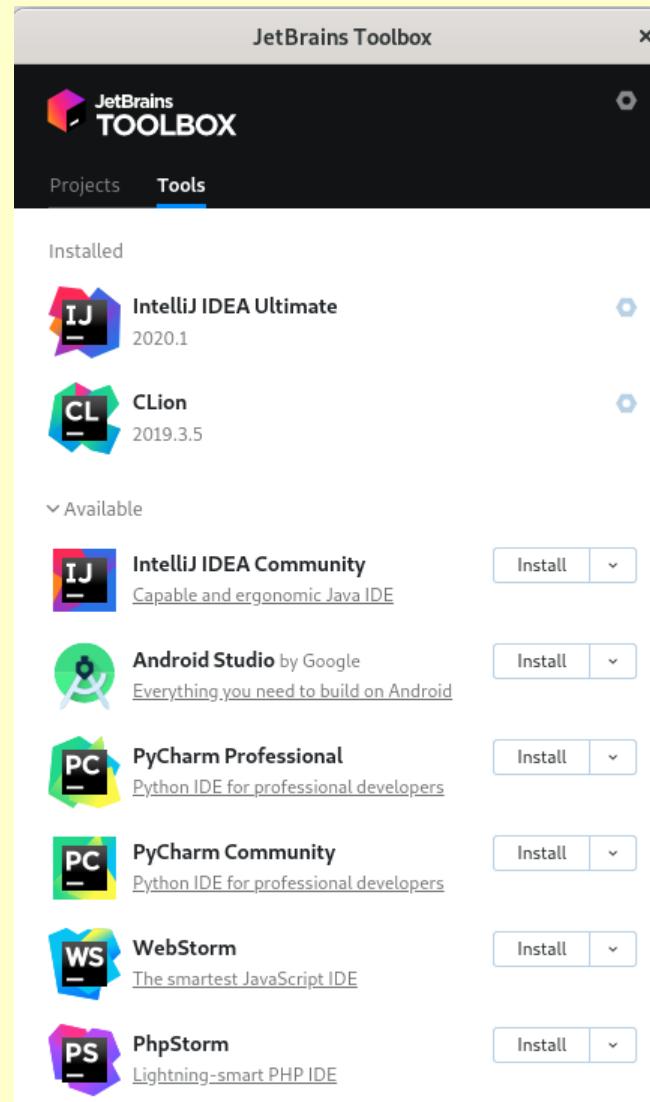
Профессиональные разработчики обязательно пользуются средой разработки.

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.

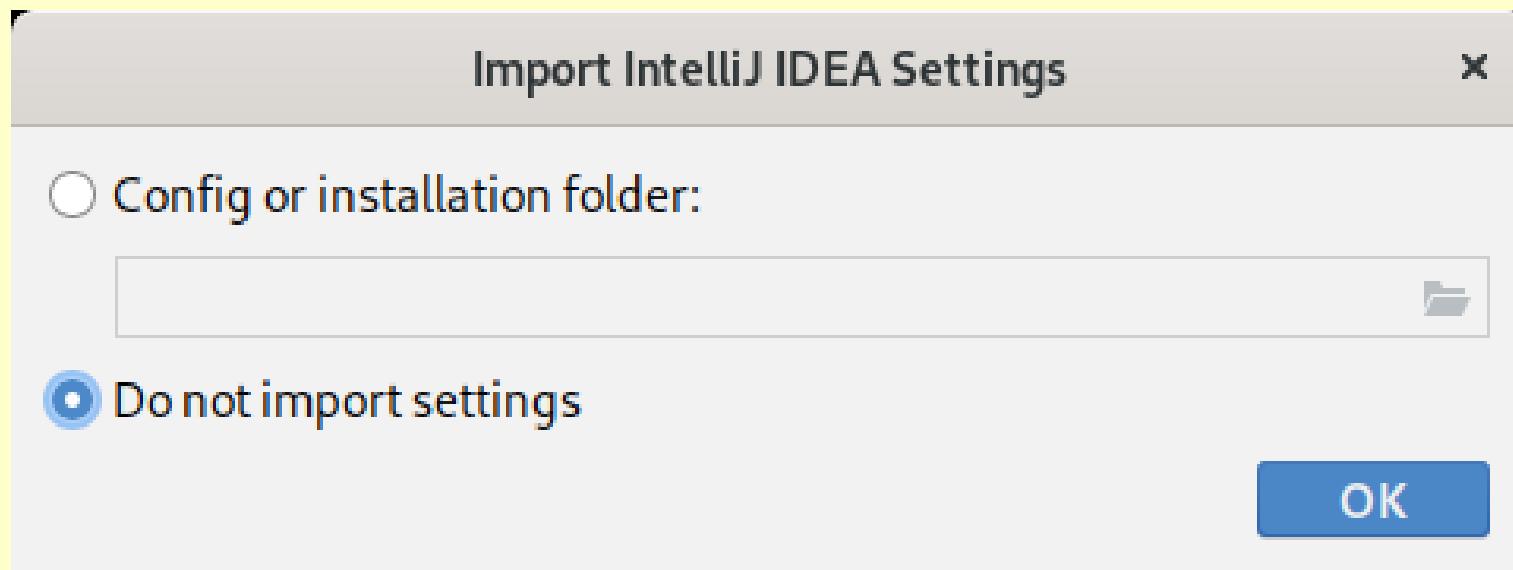
Вы можете установить Ultimate версию вместо Community. В Ultimate больше возможностей, некоторые из них полезны, но эта версия имеет больший размер и в ней больше пунктов меню, поэтому она может пугать своим перегруженным внешним видом. Кроме того, для версии Ultimate вам придется получить на сайте jetbrains студенческую лицензию, чтобы пользоваться IDEA бесплатно.

При необходимости обновить среду разработки в будущем, открывайте toolbox и нажимайте “upgrade”.

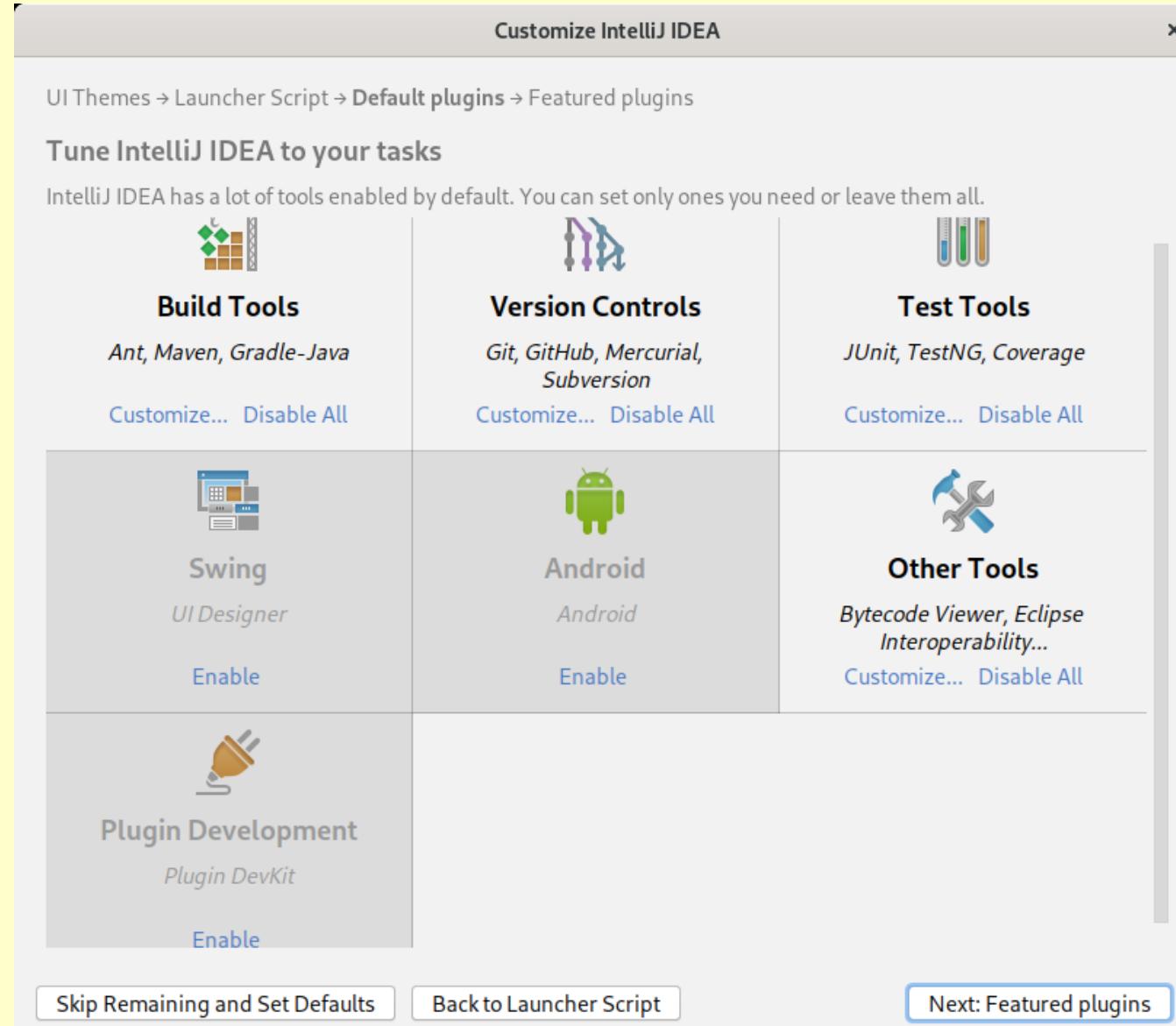
Первый запуск IDEA

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:

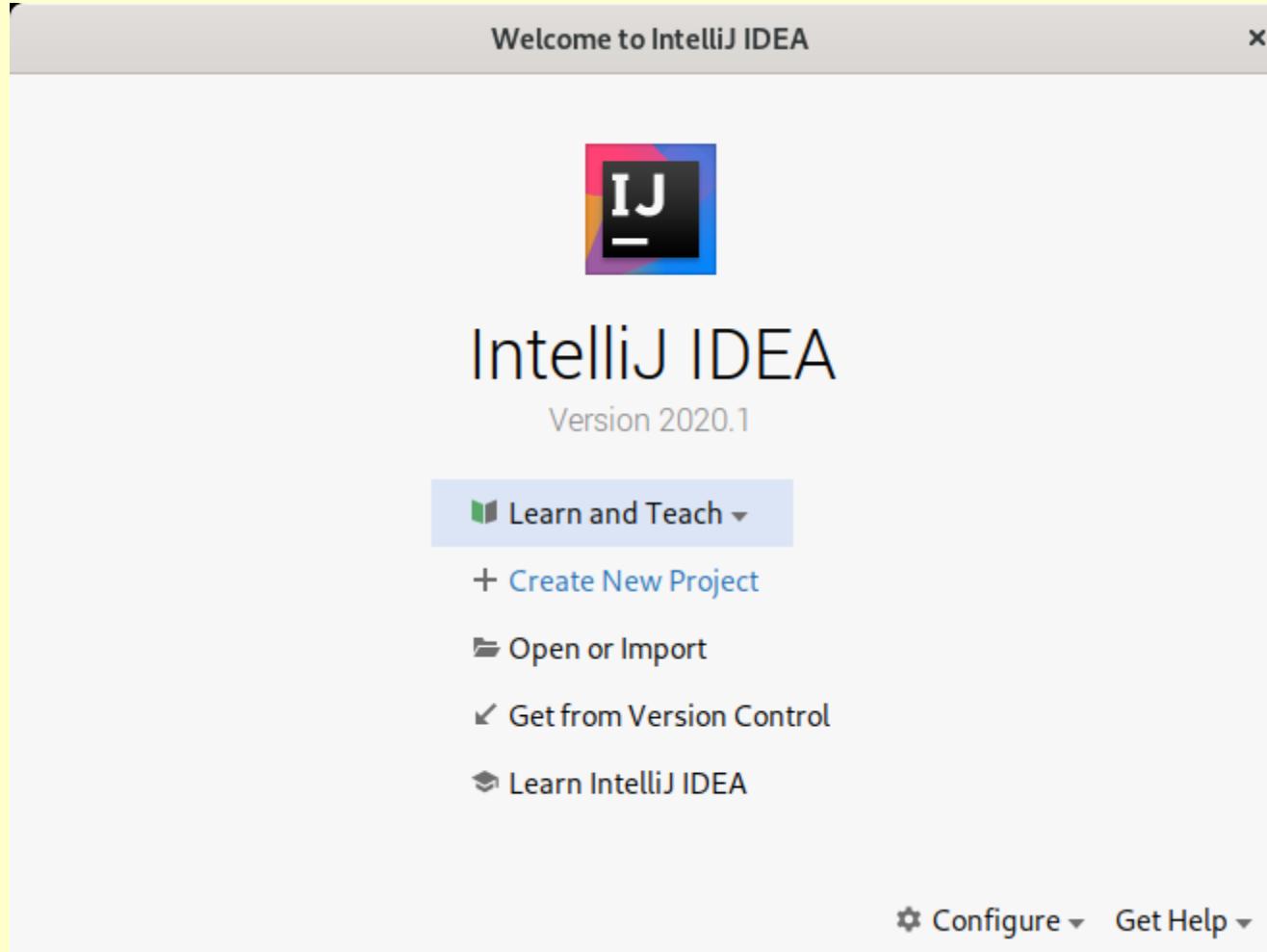


В нем отключите плагины, как я показал на картинке. Плагины лучше отключать, потому что, чем их больше, тем дольше запускается IDEA, и тем больше разных отвлекающих пунктов в меню.

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.

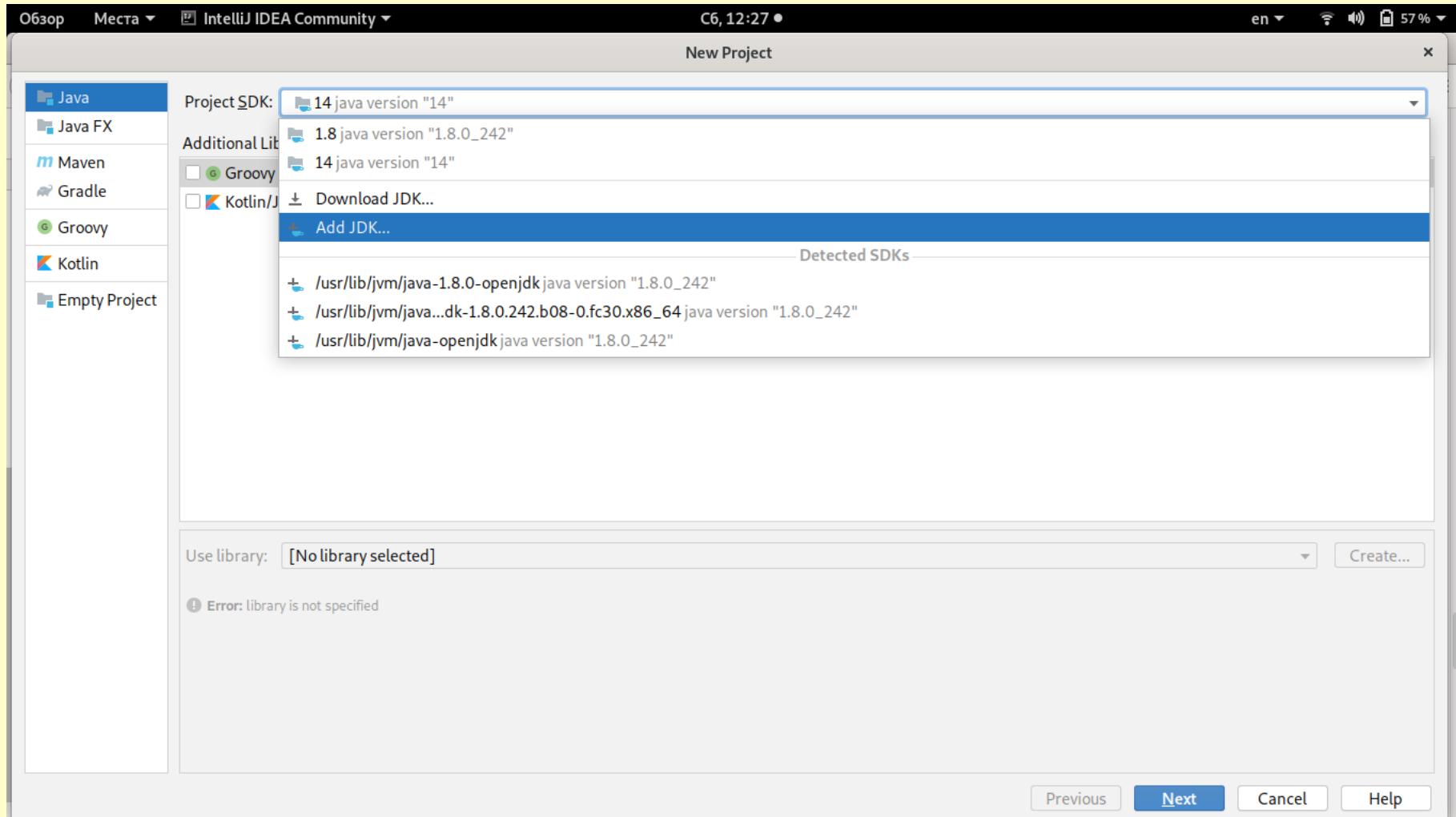
Создание проекта

Настройка закончена, после запуска IDEA вы увидите:
screenshot

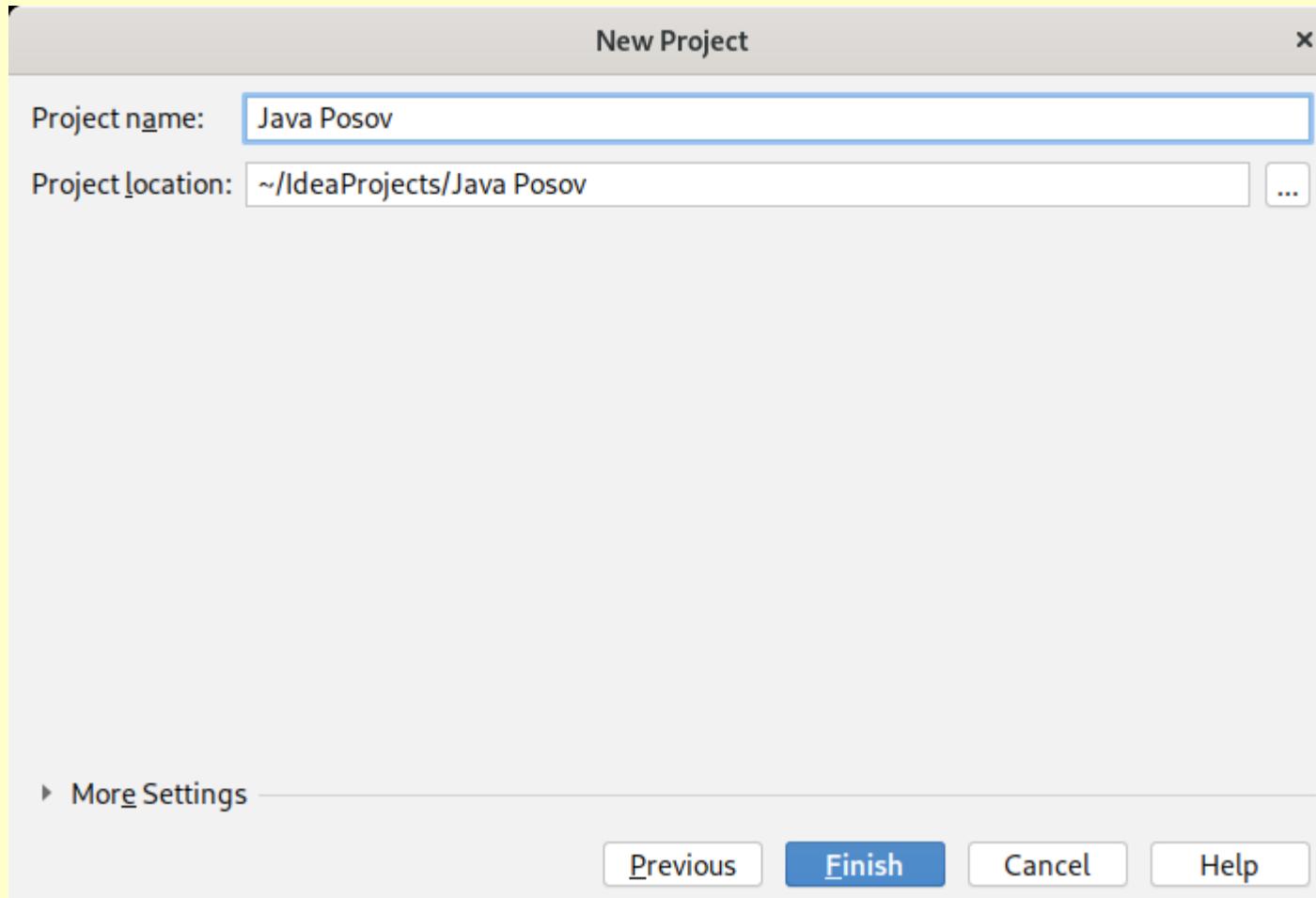


Нажмите Create New Project для создания нового проекта. Если у вас не будет начального окна, как сейчас, создать новый проект всегда можно через меню File.

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:



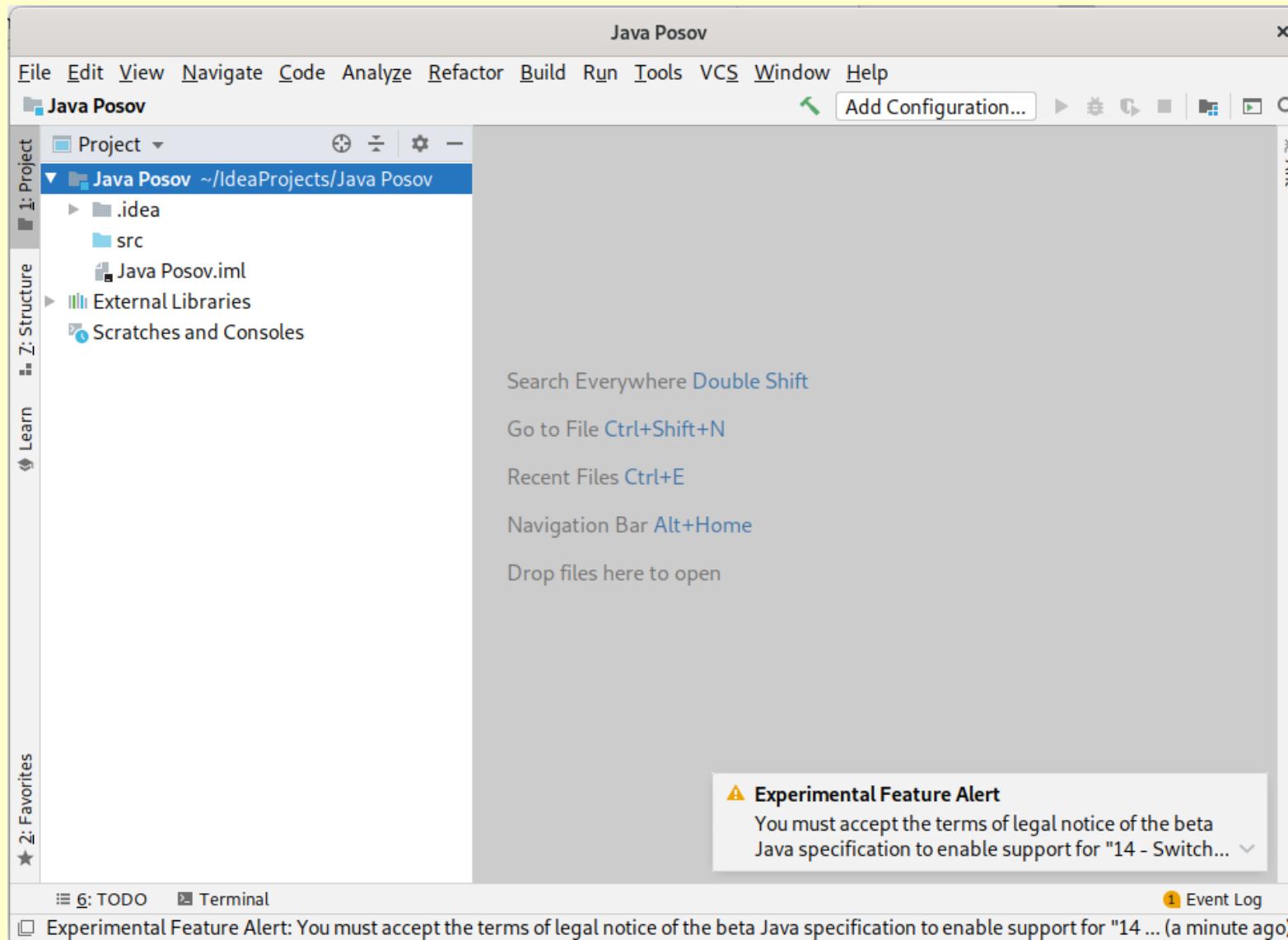
Придумайте название проекту. Там вы можете написать номер семестра, слово Java, еще какие-то логичные слова. После этого выбирайте расположение. Это папка, в которой будет находиться ваш проект. Важно:

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 menu bar includes File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, Help, and a user name va Posov. The title bar says Java Posov - HelloWorld.java. The main window has a Project view on the left showing a file named HelloWorld.java. The code editor shows the following Java code:

```
1 public class HelloWorld {  
2     public static void main(String[] args){  
3         System.out.println("Hello World!");  
4     }  
5 }
```

The line "System.out.println("Hello World!");" is highlighted. Below the code editor is a Run panel with the following information:

- Run: HelloWorld
- Output: /home/ilya/opt/jdk14/bin/java -javaagent:/home/ilya/.local/share/Java Posov/lib/ja... Hello World!
- Status: Process finished with exit code 0

At the bottom, there are tabs for TODO, Run, Messages, Terminal, and Event Log. A status bar at the bottom right indicates: Build completed successfully in 6 s 854 ... (moments ago) 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.

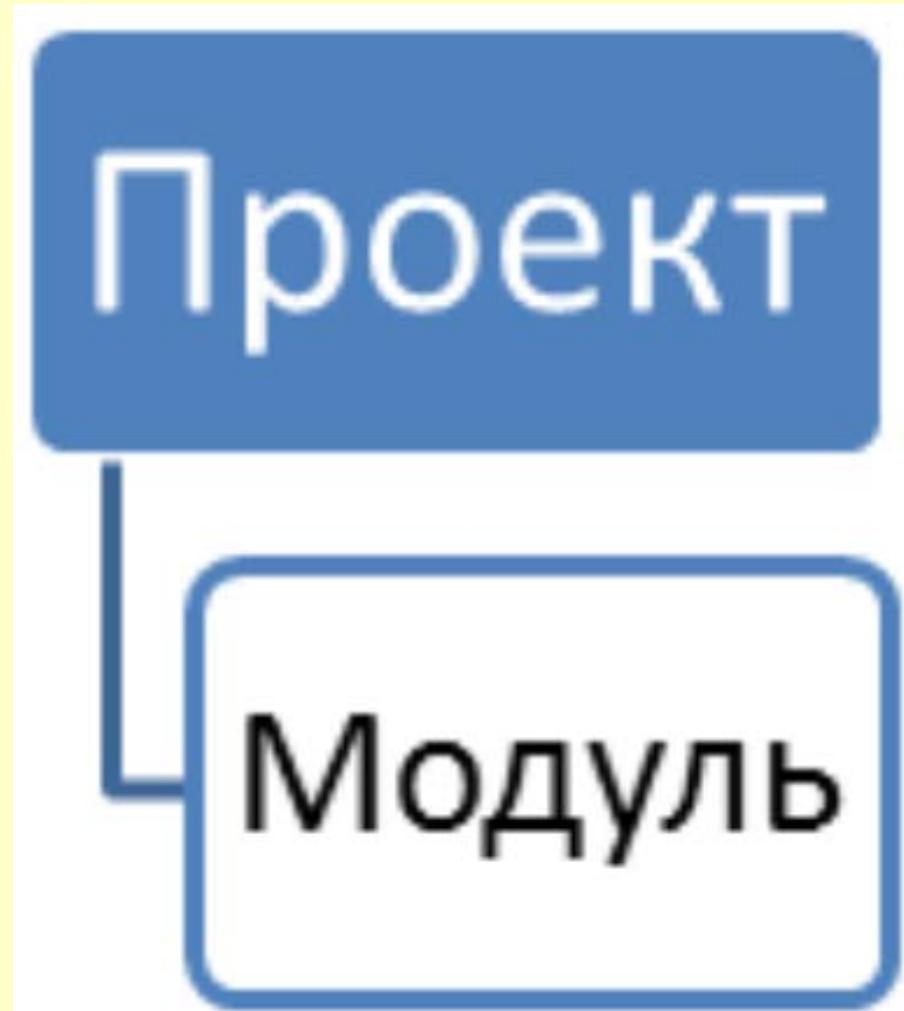
Теперь наконец-то мы можем создать наше первое приложение и посмотреть, как оно работает.

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.

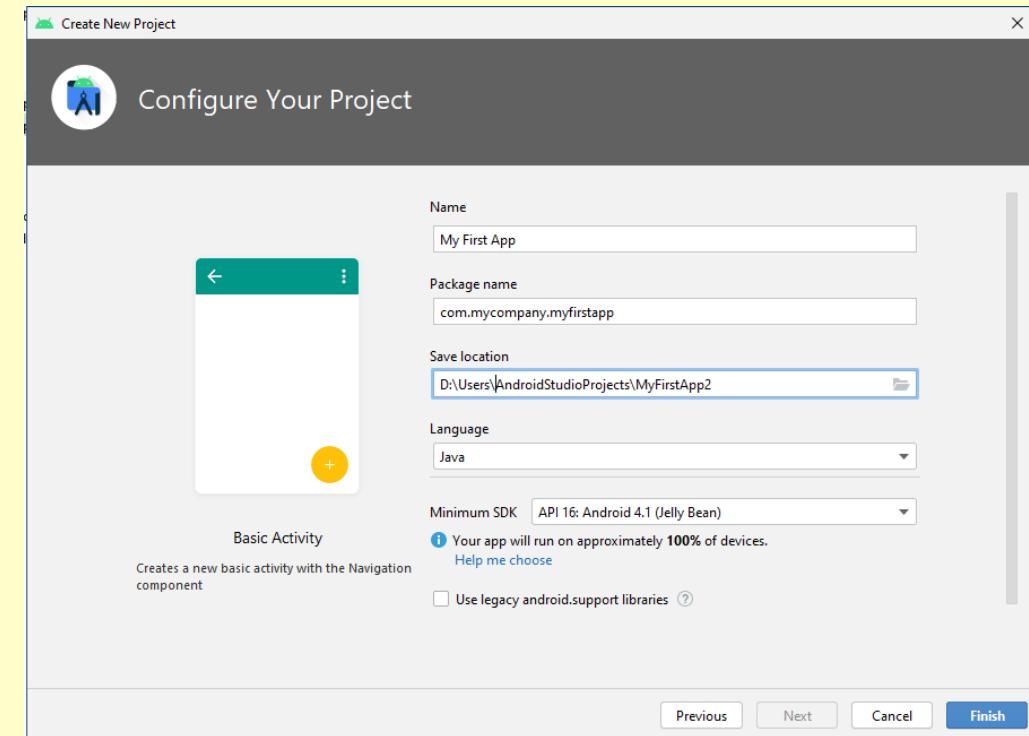


Рисунок 1. Создание нового проекта в Android Studio

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.

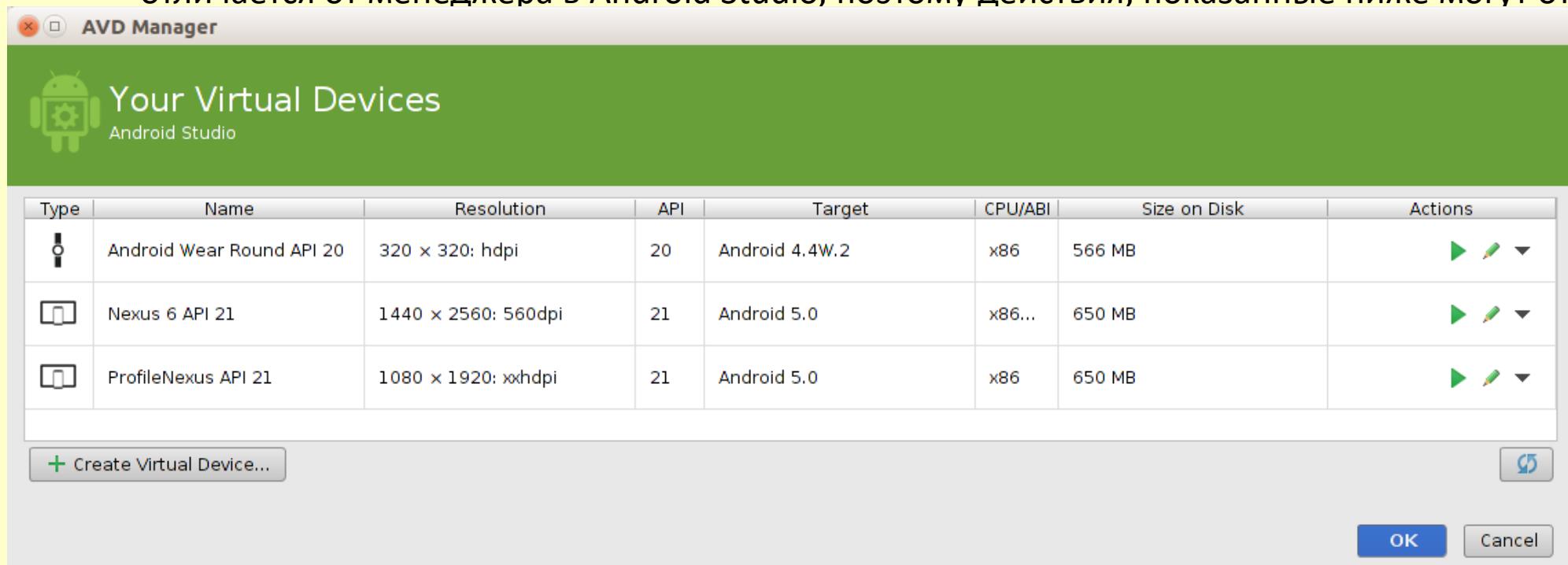


Рисунок 2. Менеджер виртуальных устройств

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`.

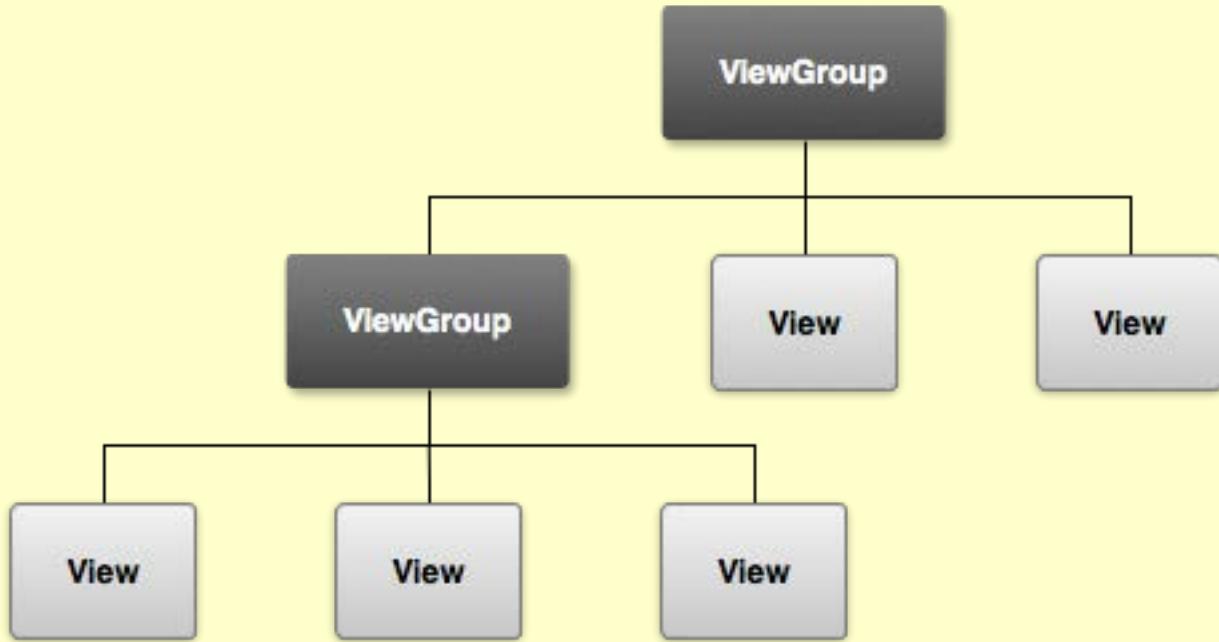


Рисунок 3 .ViewGroup позволяет строить дерево объектов разметки.

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

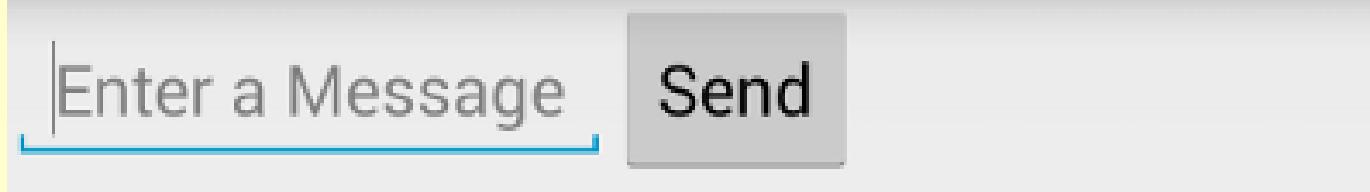


Рисунок 2.Кнопка и поле вводе с шириной равной "wrap_content".

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.

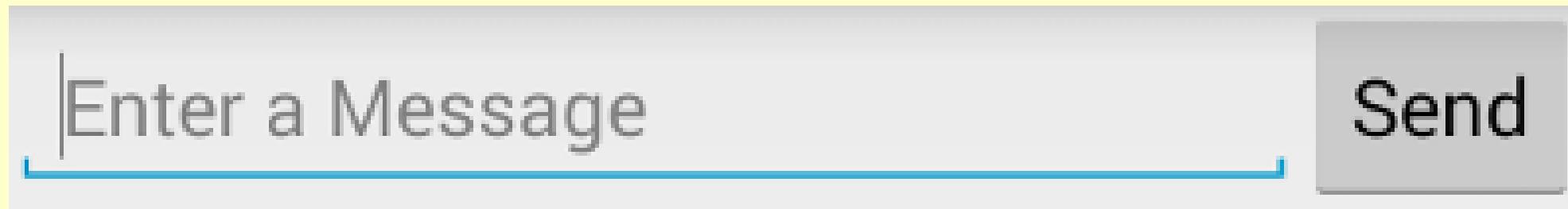


Рисунок 3. Вид поля ввода при установленном параметре weight.

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]:	10.0.2.4/24	①
IP4.GATEWAY:	10.0.2.1	②		
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     www          # 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-Original-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

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          D         Θ Thu Jan  7 20:57:36 2016
Dd596.jpg          D         Θ Thu Jan  7 20:57:36 2016
Dd680.jpg          ...       ...
Dd681.jpg          A 4494092 Sun Feb 13 16:24:04 2011
Dd680.jpg          A 3842680 Sun Feb 13 16:14:56 2011
Dd681.jpg          A 4087313 Sun Feb 13 03:10:45 2011
Dd680.jpg          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: 8fa415d163cf0496576a136652cac2f6b/7935058721407822980\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 -n -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 Nov 2013 10:24:00 +0000\r\nContent-Type: image/png\r\nContent-Length: 1382\r\n\r\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.

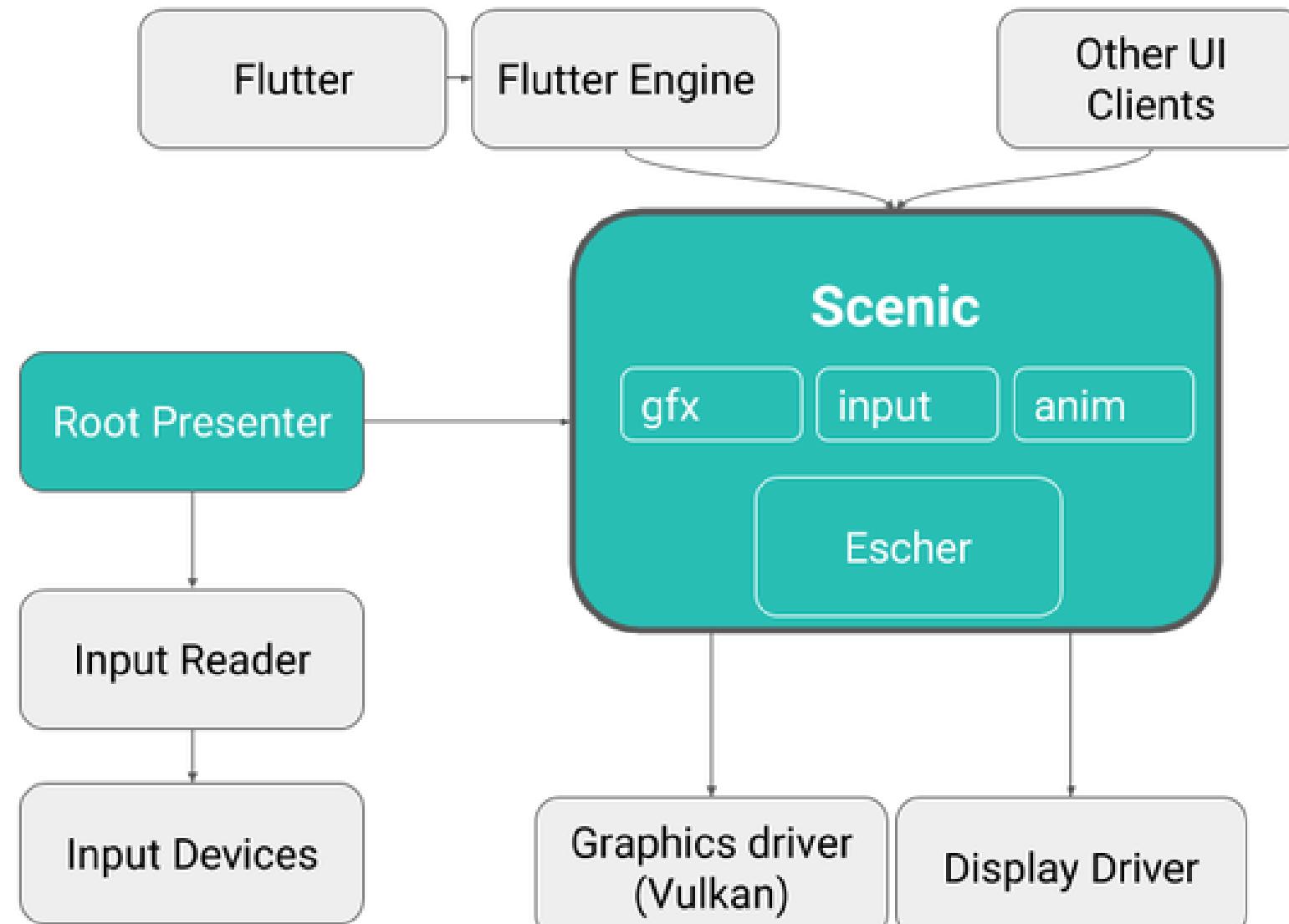
Загружаются они при помощи процесса devhost, а управляются менеджером устройств (devmg, Device Manager).

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»:

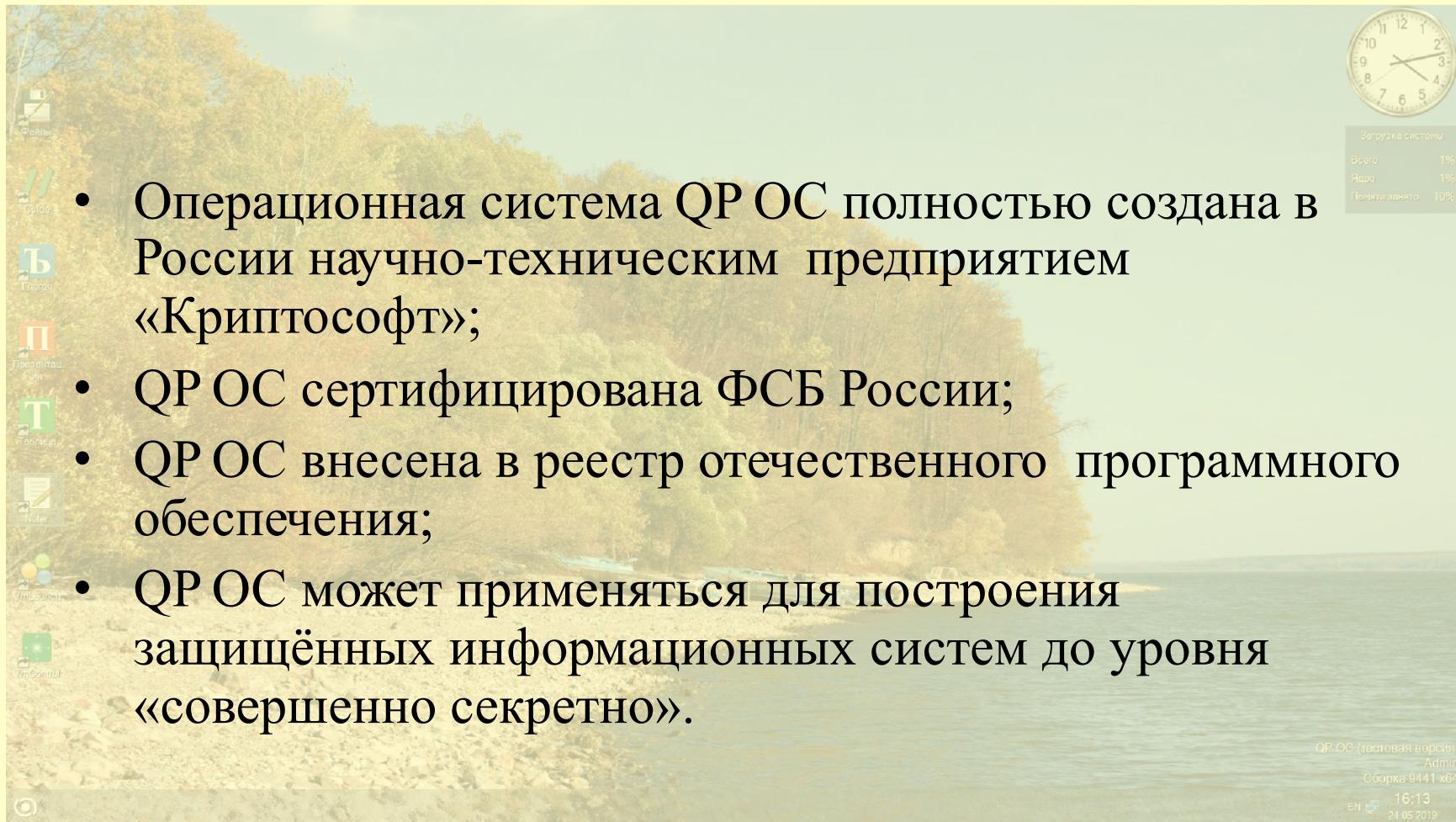
Преимущества использования полностью отечественной операционной системы

- 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.





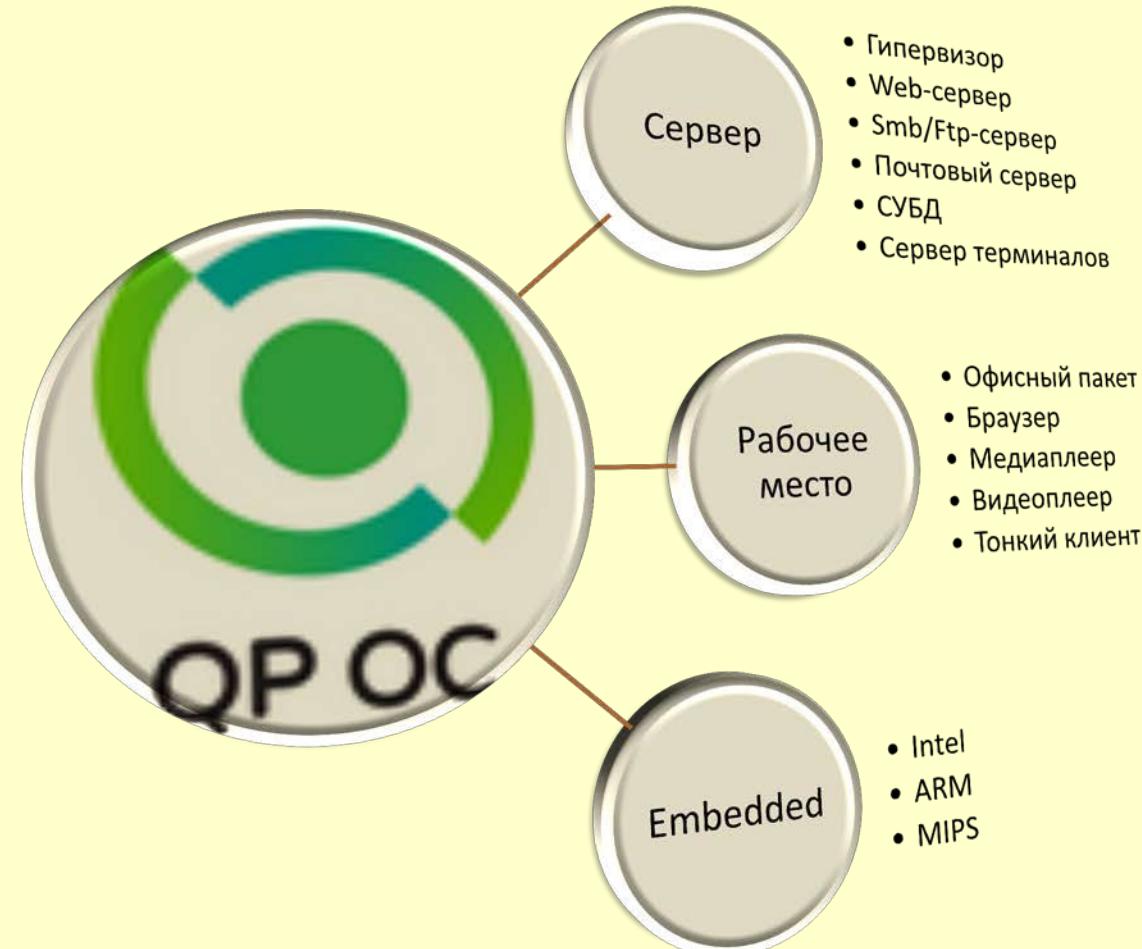
QP ОС



- Операционная система QP ОС полностью создана в России научно-техническим предприятием «Криптософт»;
- QP ОС сертифицирована ФСБ России;
- QP ОС внесена в реестр отечественного программного обеспечения;
- QP ОС может применяться для построения защищённых информационных систем до уровня «совершенно секретно».



Применение QP OC





Полный комплекс механизмов безопасности



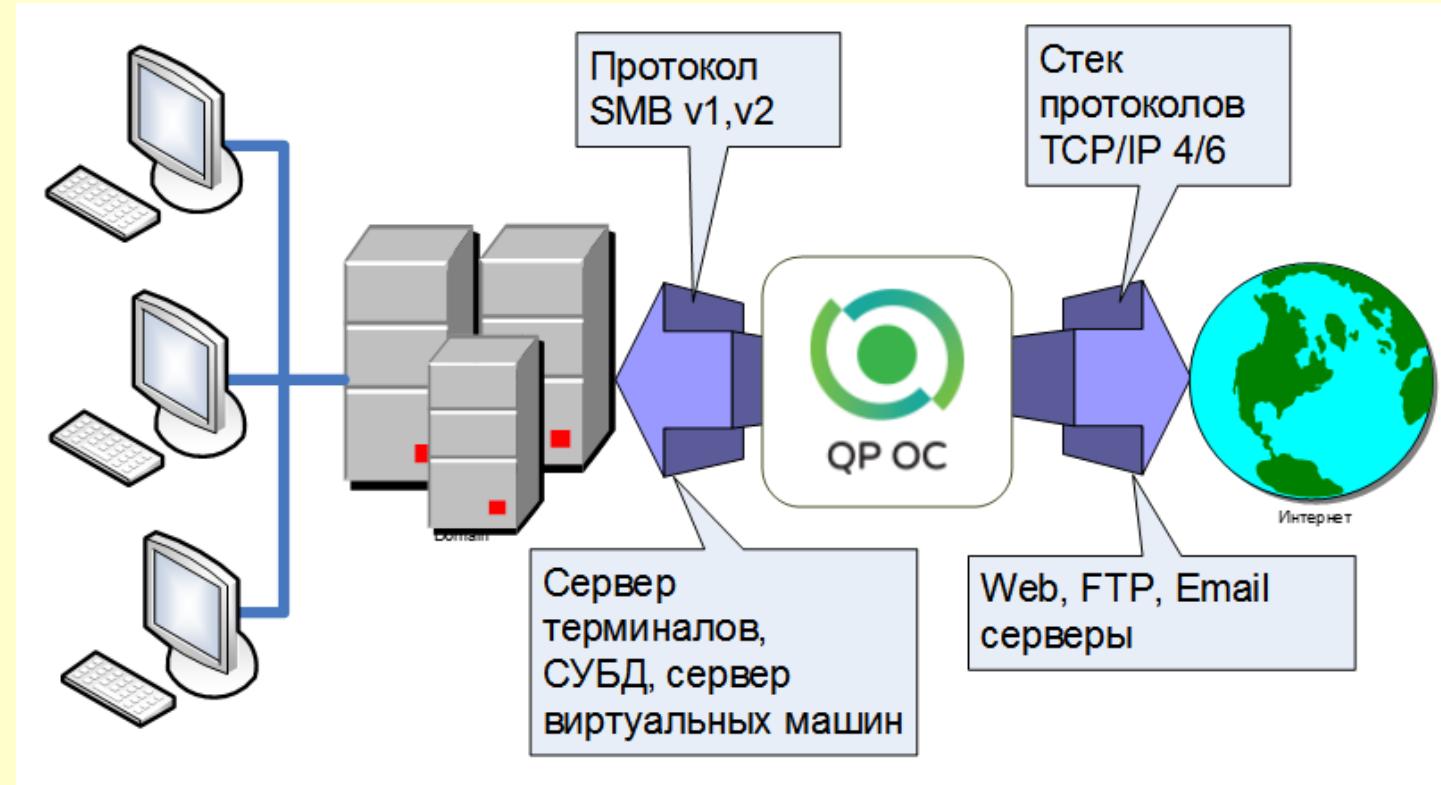


Серверные компоненты в составе QP ОС





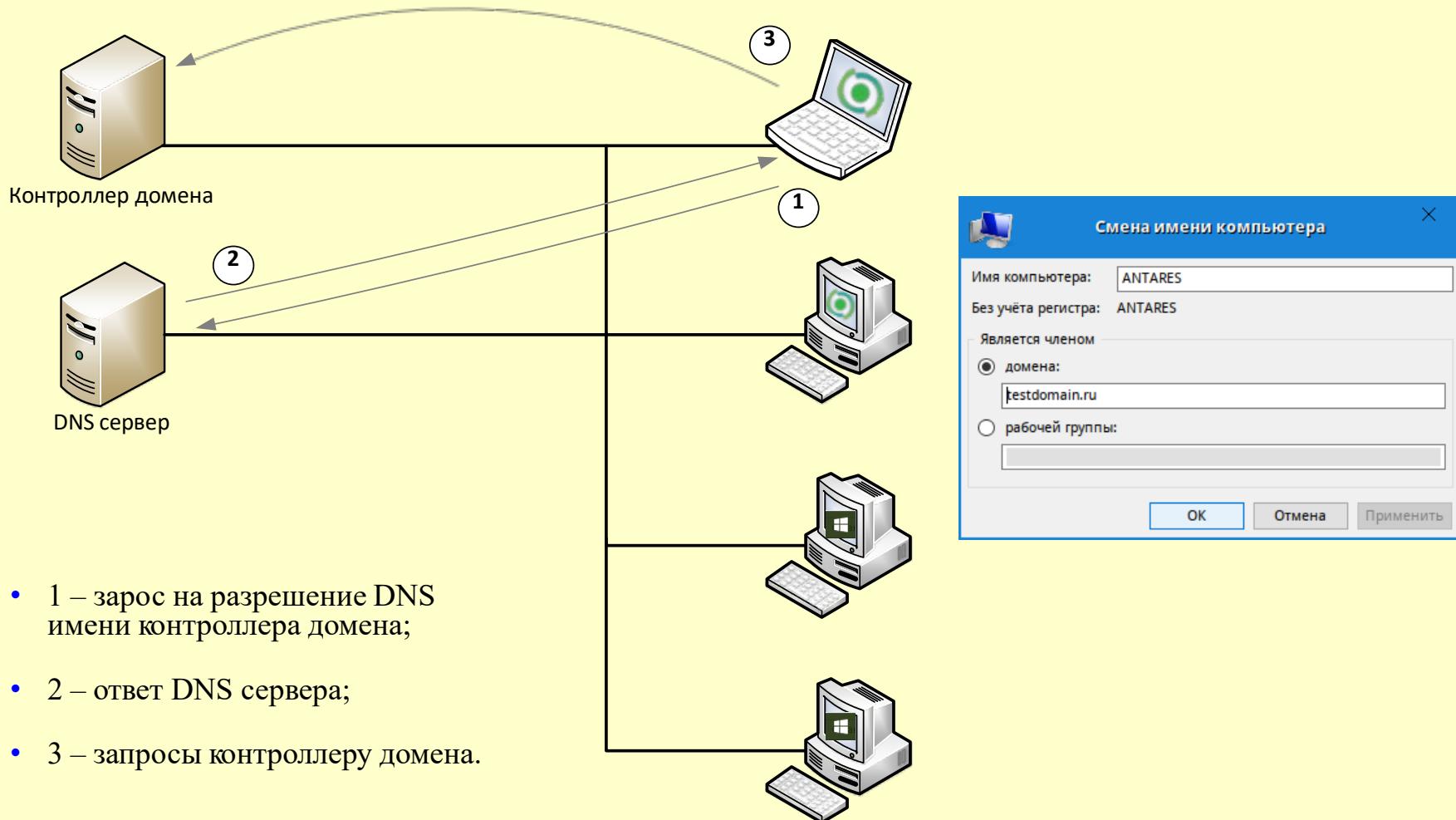
Сетевые возможности QP ОС



- 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

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 downwards. 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

AF:47 =Q23+Q24+Q25+Q26+Q27+Q28+Q29+Q30+Q31+Q32+Q33+Q34+Q35+Q36+Q37+BW23+BW24

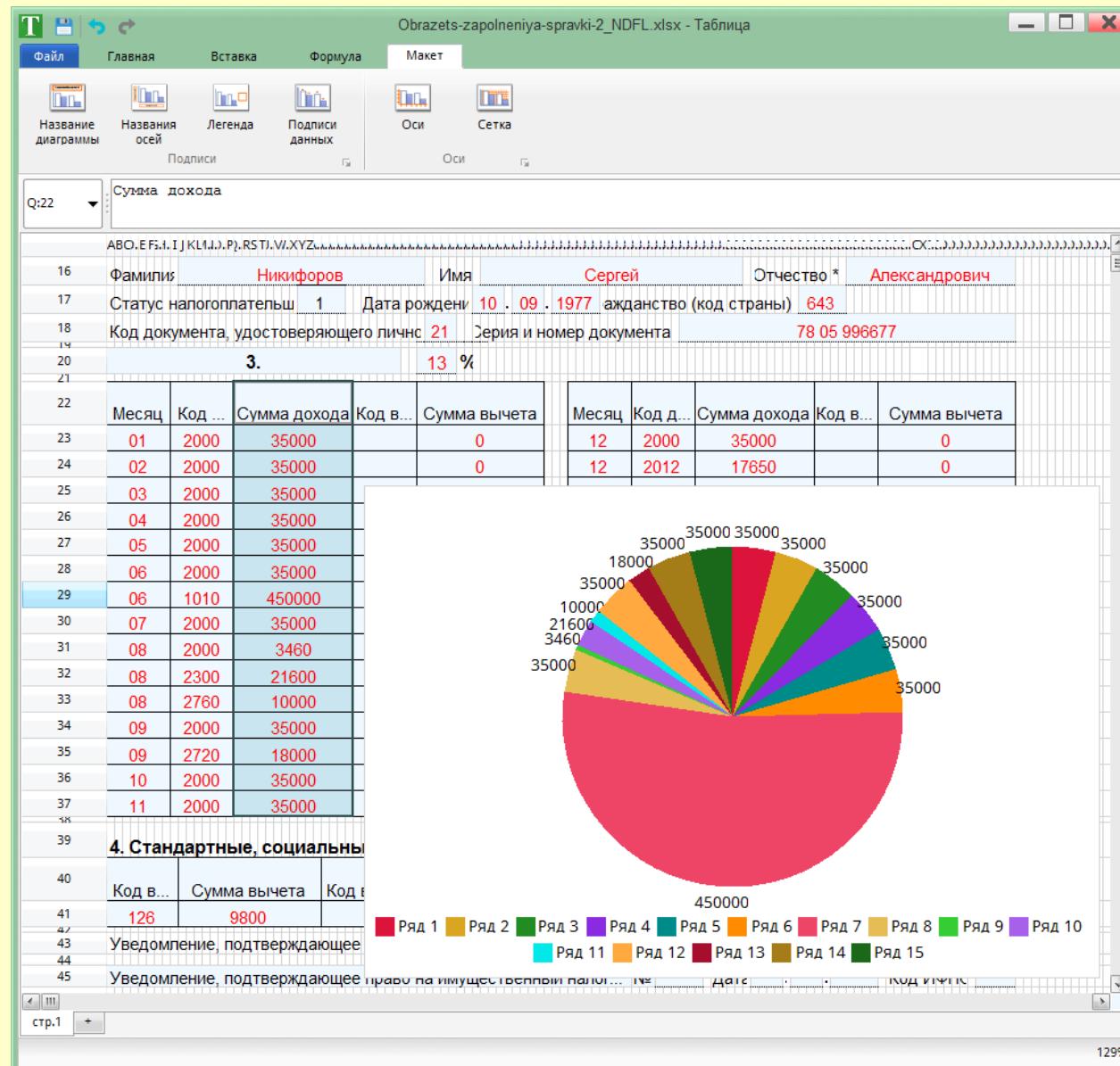
22	Месяц	Код ...	Сумма дохода	Код в...	Сумма вычета	Месяц	Код д...	Сумма дохода	Код в...	Сумма вычета
23	01	2000	35000		0	12	2000	35000		0
24	02	2000	35000		0	12	2012	17650		0
25	03	2000	35000		0					
26	04	2000	35000		0					
27	05	2000	35000		0					
28	06	2000	35000		0					
29	06	1010	450000		0					
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 Катерины II. 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 «апельсиновую корку» с бедер. Кино Пропустившая спецвыпуск «Голоса» дочь Алсу высказалась о суде над отцом.

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?

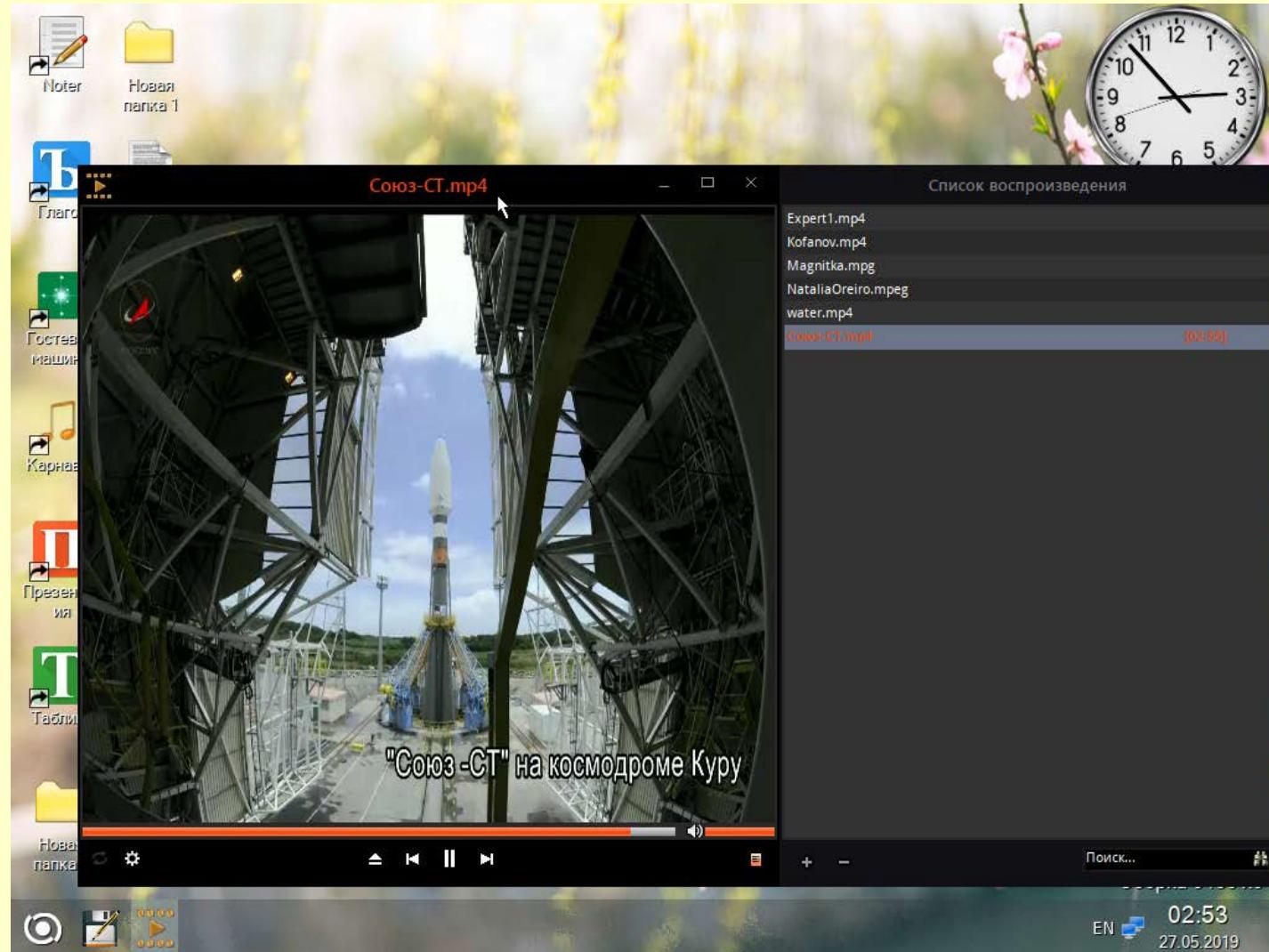
Помогите нам сделать государственные сайты лучше, это займет всего несколько минут

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 Добавка 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.

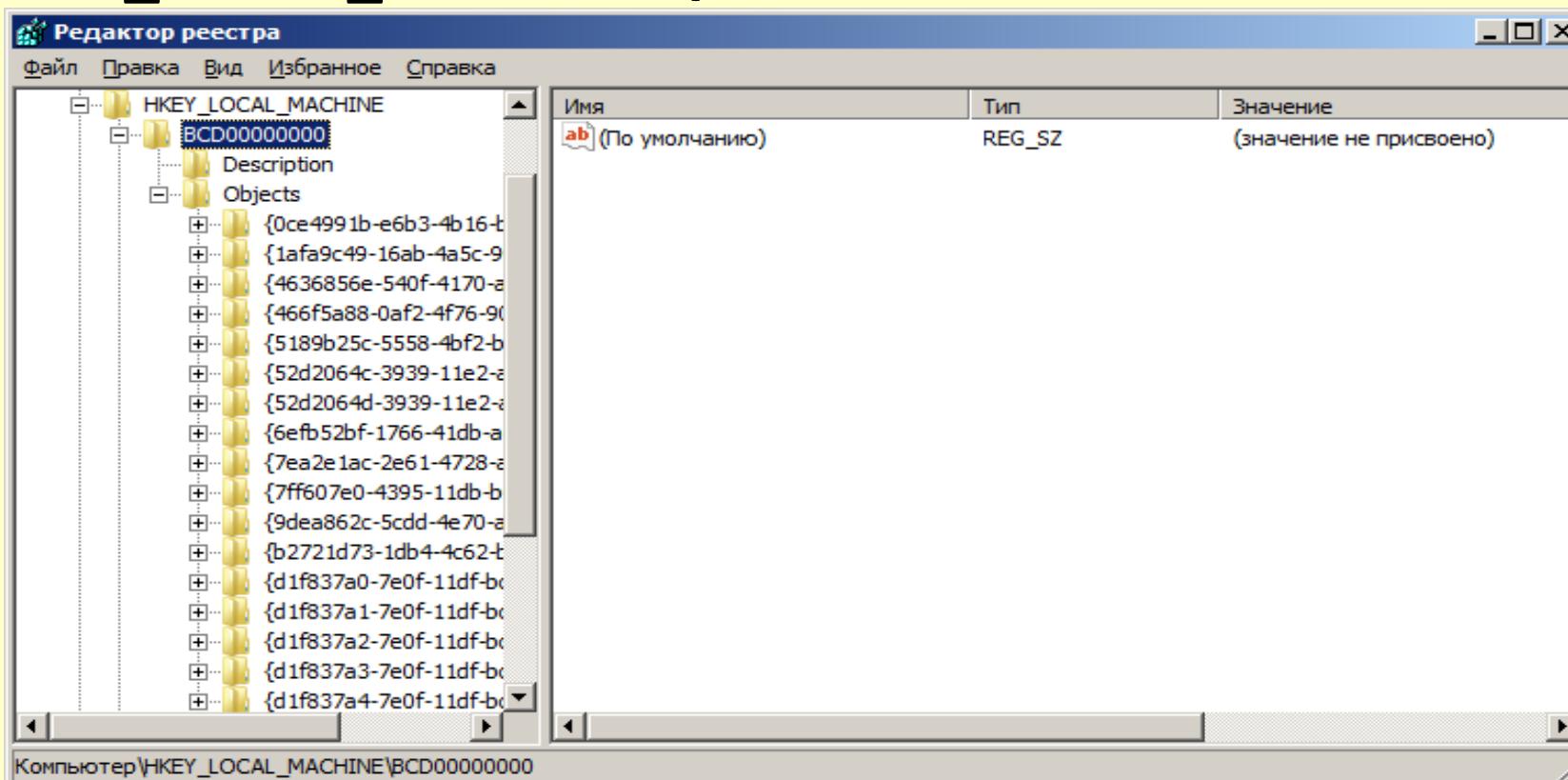
В Windows 7 разделы реестра SAM и SECURITY отображаются , однако не отображается их содержимое, для просмотра которого можно воспользоваться аналогичным приемом.

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.

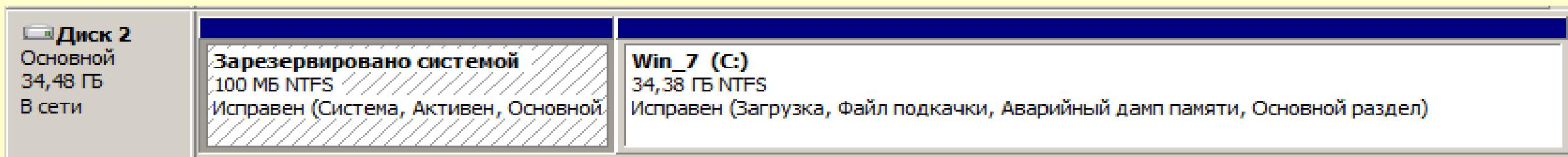
Нарушение целостности файлов реестра (нарушение структуры данных) или неверное значение отдельных критических параметров может привести к краху системы . Поэтому, прежде чем экспериментировать с реестром, позаботьтесь о возможности его сохранения и восстановления.

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.